INFORMATIONAL INTEGRATION OF PRODUCT DEVELOPMENT SOFTWARE IN THE AUTOMOTIVE INDUSTRY

THE ULEO APPROACH

♦

DISSERTATION

to obtain
the doctor's degree at the University of Twente,
on the authority of the rector magnificus,
prof. dr. W.H.M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Friday, April 29, 2005, at 13.15 hours

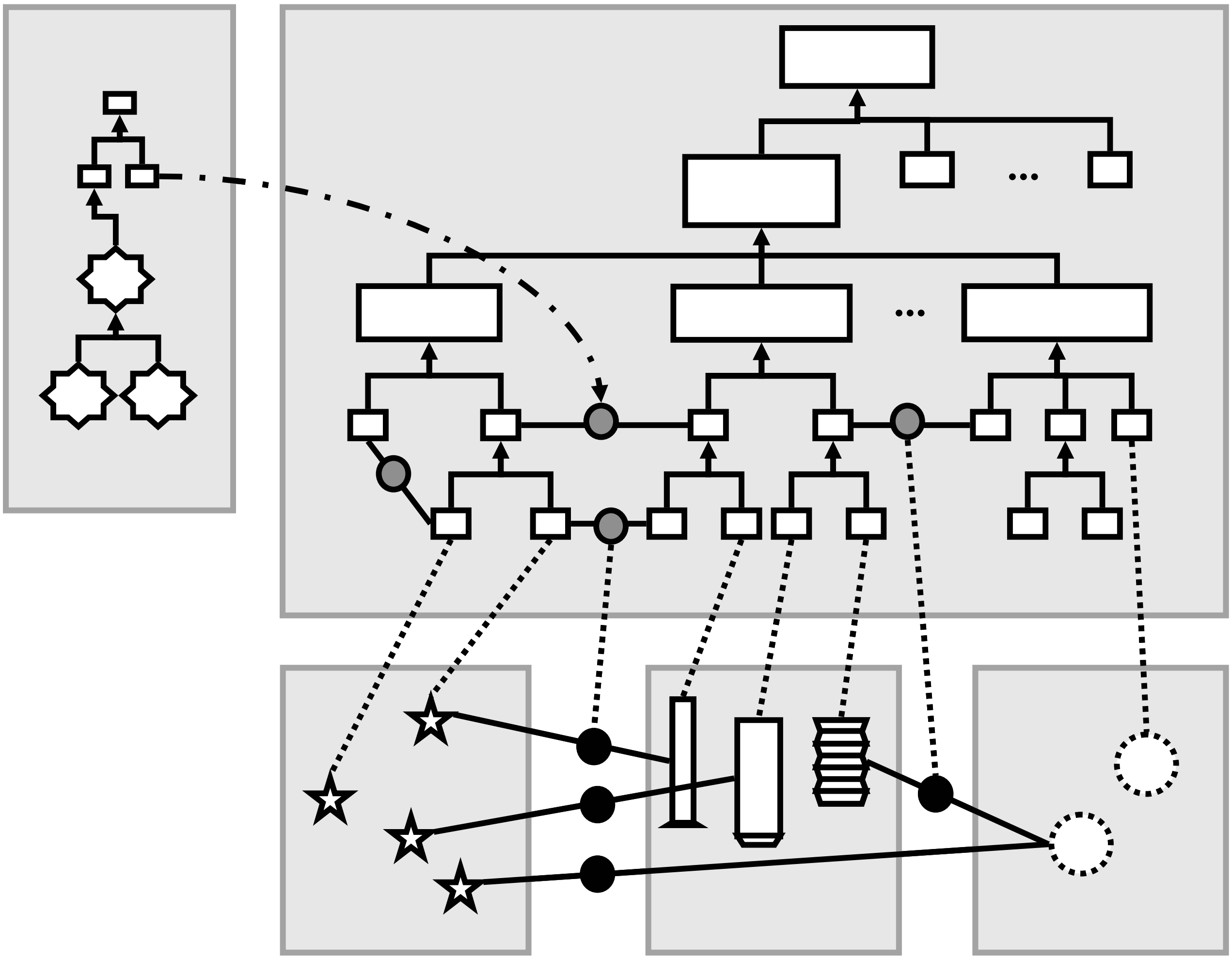
by
Johann Ulrich Zimmermann
born on July 4, 1962
in Ulm on the Danube, Germany

This dissertation has been approved by the promotor
prof. dr. ir. F.J.A.M. van Houten.

Johann U. Zimmermann

Informational Integration of Product Development Software in the Automotive Industry

The ULEO Approach



Title: Informational Integration of Product Development Software
in the Automotive Industry
Subtitle: The ULEO Approach
Author: Johann Ulrich Zimmermann
Contact: DaimlerChrysler Research, Dept. of Product and Production Modeling, REI/IP
P.O. Box 2360, 89013 Ulm, Germany, *Johann.Zimmermann@DaimlerChrysler.com*
Date: 2005-04-29

Categories: Ph.D. Thesis; Doctoral Thesis; Dissertation; University of Twente, the Netherlands

ISBN: 90-365-2164-5
This thesis is distributed in a limited edition by the Laboratory of Design, Production and Management, Dept. of Engineering, University of Twente with the ISSN 1386-5307.

“I know that I don’t know anything – and even that I hardly know”

(Socrates, Greek philosopher, ca. 470 to 399 B.C.)

This thesis reports on a research work I felt to be especially well balanced in terms of goals and realization. This feeling originates from its organizational embedding into both the university environment at the *Department of Engineering* of the University of Twente and the industry-focused *Product and Production Modeling Department* at DaimlerChrysler Research and Technology. On this fruitful ground, scientific challenges and practical needs fused to form the goals I pursued in this work. Later, this ground enabled me not only to discuss possible solutions at a high scientific level but also to verify them in hot-blooded discussions with experts from automotive practice. Finally, I was lucky to find a first field for the practical implementation and productive deployment of my ideas.

But – why are you doing this? Why do you strive for a higher degree at all? As this question is certainly a very personal one, I can naturally only answer for myself, starting with a kind of negative list: I do not expect significant material benefits or jumps in my career; I do not expect an enhanced reputation; I also do not expect to raise my self-esteem. No, the answer can be found somewhere in the field of my personal values: it is related to my personal idea of the meaning of life. I consider erudition as a value of its own that is – quite generally but also practically visible – able to prevent the evil and promote the good. And based on this motivation, I am thankful to all those who made this research work possible in such a challenging context or who supported me in carrying it out or who helped to bring it to a good end. As with any acknowledgements, there is a danger of not mentioning someone who should definitely have been named. I therefore wish to generally refrain from acknowledging individual persons to whom I could express my gratitude. “Generally” means …

… with the exception of the two persons most tightly involved: I would like to sincerely thank Prof. dr. ir. F.J.A.M. van Houten and Dr.-Ing. Siegmar Haasis for their commitment in enabling and supporting this work;

… and with the exception of the only persons who suffered from my work. Therefore, I would like to put down in writing special thanks to my family: to my wife, Doris, and our two little daughters, Laura and Helen, and to my mother, Anneliese, who often had to do without one of the most valuable things I can give them – time.

Diese Dissertation berichtet über eine Forschungsarbeit, die ich als besonders ausgewogen empfand im Hinblick auf ihre Ziele und ihre praktische Durchführung. Der Grund für diese Ausgewogenheit war die organisatorische Einbettung dieser Arbeit, die zum einen geprägt wurde durch das universitäre Umfeld des *Department of Engineering* an der Universität Twente, und zum anderen durch die Abteilung *Produkt- und Produktionsmodellierung* der industrieorientierten DaimlerChrysler Forschung & Technologie. Auf diesem fruchtbaren Boden verschmolzen wissenschaftliche Herausforderungen und praktische Anforderungen und führten schließlich zu den Zielen, die ich in dieser Arbeit verfolgte. Auf dieser Grundlage konnte ich dann im weiteren Verlauf der Arbeiten identifizierte Lösungsansätze auf hohem wissenschaftlichem Niveau diskutieren, sie aber auch gleichzeitig in kontroversen Debatten mit Experten aus der Automobilentwicklung evaluieren. Schließlich gelang es mir, ein erstes Feld für die praktische Umsetzung und produktive Nutzung meiner Konzepte zu finden.

Aber – warum streben Menschen überhaupt nach einem akademischen Grad? Da diese Frage sicherlich sehr persönlicher Natur ist, kann ich naturgemäß nur für mich selbst antworten und möchte dazu mit einer Art von Negativliste beginnen: ich erwarte weder bedeutende materielle Verbesserungen noch Karrieresprünge, genauso wenig wie ein höheres Ansehen; und ich werde damit auch mein Selbstwertgefühl nicht steigern. Nein, die Antwort liegt vielmehr irgendwo im Bereich meiner persönlichen Werte: sie hat etwas zu tun mit meiner Vorstellung vom Sinn des Lebens. Bildung ist für mich ein Wert an sich, der – ganz universell und dennoch praktisch erlebbar – Böses zu verhindern und Gutes hervorzubringen vermag. Und vor diesem Hintergrund bin ich all jenen dankbar, die diese Forschungsarbeit in einem so herausfordernden Umfeld ermöglichten, oder die mich bei ihrer Durchführung unterstützten, oder die mir halfen, sie zu einem guten Abschluß zu bringen. Alle Danksagungen laufen Gefahr, daß Menschen nicht genannt werden, die unbedingt hätten genannt werden müssen. Deshalb möchte ich im Rahmen dieser Dissertation im Grundsatz davon absehen, einzelnen Menschen meinen Dank auszusprechen. „Im Grundsatz“ heißt, …

… mit Ausnahme der beiden Personen, die am engsten an meinen Arbeiten beteiligt waren: so möchte ich gerne Herrn Prof. dr. ir. F.J.A.M. van Houten und Herrn Dr.-Ing. Siegmar Haasis herzlich danken für ihr Engagement mit dem sie meine Arbeit ermöglichten und unterstützten;

… und mit Ausnahme der einzigen Menschen, die unter dieser Forschungsarbeit zu leiden hatten. Ich möchte daher meiner Familie ganz besonders danken: meiner Frau, Doris, und meinen beiden kleinen Töchtern, Laura und Helen, und meiner Mutter, Anneliese – denn sie mußten oft ohne eines der wertvollsten Dinge auskommen, die ich ihnen geben kann – Zeit.

Meinen Eltern.
To my Parents.

Part I – INTRODUCTION

This first part of the thesis provides the reader with information that will facilitate the handling of this document and open access to the topic.

Chapter 1 Overview of this Thesis

This chapter sketches the contents and the handling of this document.

1.1 Abstracts

The following English and German abstracts provide brief insight into the major goals and solutions of the research work.

English Abstract

After an informal characterization of today’s situation in automotive product development, this thesis identifies measures for improving the information-technological support of the respective processes. “Improvement” means to raise the effectiveness and efficiency of the supported processes, to reduce superfluous product variety and to increase the product quality. This research focuses on two major goals, of which the paramount one is to supply product development engineers and their software applications with more and better information by fusing isolated isles of information. The second, subordinated objective is to automate routine work in order to avoid errors and detours and to cut the costs caused by product changes.

To realize these goals, a new approach termed *Universal Linking of Engineering Objects (ULEO)* is proposed. ULEO's mission is to open up all the software applications deployed in a company’s product development for sharing information and to embed them into a common *Global Information Space (GIS)*. This includes supplier software. Inside the GIS, all the relevant information is accessible to all interested parties online. The information may be expressed in any desired terminology, not fixed at the compile time of the communicating applications. Unlike existing solutions such as *Semantic Web* compatibles, ULEO is tailor-made to engineering. To achieve the stated goals, ULEO specifies (1) appropriate basic information types within the GIS, (2) an optimized and balanced representation formalism, (3) a set of online services supporting information and control flow in the GIS, (4) a software architecture, and (5) specifications for applying ULEO.

Each application may access information about the kind and meaning of other applications’ information online from a virtual *Integrated Information Model (IIM)*. Starting from *semantic kernels*, an application may utilize sophisticated relations to navigate through the GIS. It may also retrieve the corresponding instance information stored inside application-specific and proprietary product, process, and resource models, as well as the newly provided relations that correlate them across applications (PPR backbone). Automation strategies and other company know-how can be shared in the GIS by storing them inside the IIM with the applications interpreting this information at runtime.

As the ULEO concepts have been adopted by a major automotive OEM for use in productive product development, the *ULEO server* software prototype has been further enhanced and productively applied since early 2005. The conclusions of this thesis therefore also deal with pertinent experiences and validations.

Keywords not appearing in the title. Features, global information space, semantic integration.

Kurze Zusammenfassung

Nach einer informellen Charakterisierung der heutigen Situation im Bereich der automobilen Produktentwicklung identifiziert diese Dissertation Maßnahmen zur verbesserten informationstechnischen Unterstützung der beteiligten Prozesse. Die betrachteten Verbesserungen erhöhen die Effektivität und Effizienz dieser Prozesse. Darüber hinaus verringern sie eine unnötige Produktvielfalt und steigern die Produktqualität. Diese Forschungsarbeit fokussiert im wesentlichen zwei Hauptziele, von denen die Versorgung von Produktentwicklungsingenieuren und -Software mit mehr und besserer Information im Vordergrund steht. Damit werden Informationsinseln verschmolzen. Das zweite, untergeordnete Ziel dieser Arbeit ist die Ermöglichung einer Automatisierung von Routinearbeiten in der Produktentwicklung. Dadurch lassen sich die Häufigkeit von Fehlern und umständlichen Lösungen verringern und aufgrund von Produktänderungen entstandene Kosten reduzieren.

Zur Erreichung der genannten Ziele wird der Lösungsansatz *Universal Linking of Engineering Objects (ULEO)* entwickelt. Er ermöglicht es, prinzipiell beliebige Software-Anwendungen aus dem Bereich der Produktentwicklung für den Fluß von Information zu öffnen und sie in einen *Globalen Informationsraum (GIS)* einzubetten. Dies gilt auch für Software, die von Zulieferfirmen eingesetzt wird. Innerhalb des GIS steht die gesamte Information allen interessierten Anwendungen online, also per Interprozeßkommunikation, zur Verfügung. Dabei darf die Information auf einem Wortschatz basieren, der zum Zeitpunkt der Erstellung der kommunizierenden Anwendungen noch nicht bekannt war. Im Unterschied zu existierenden Lösungen, wie etwa solche, die das *Semantic Web* unterstützen, ist ULEO auf den Bereich der Produktentwicklung zugeschnitten. Um die beschriebenen Ziele zu erreichen, spezifiziert ULEO (1) geeignete grundlegende Informationstypen für den GIS, (2) einen optimierten und ausgewogenen Repräsentationsformalismus, (3) eine Menge von Diensten, die den Informations- und Kontrollfluß zwischen den laufenden Anwendungen unterstützen, (4) eine Software-Architektur, und (5) Regeln zur Anwendung von ULEO.

Jede Software-Anwendung kann im GIS auf Information zugreifen, die die Art und Bedeutung der von anderen Anwendungen verarbeiteten und angebotenen Information beschreibt. Diese Information ist online aus einem virtuellen *Integrierten Informationsmodell (IIM)* abrufbar. Ausgehend von *semantischen Kernen*, kann eine Anwendung entlang von leistungsfähigen Relationen durch den GIS navigieren. Sie kann darüber hinaus auch auf Instanzinformation zugreifen, die in anwendungsspezifischen, proprietären Datenbanken gespeichert ist, und auch auf die neu hinzugekommenen Relationen, die solche Produkt-Prozeß-Ressource-Information vernetzen. Die Gesamtmenge solcherlei vernetzter Instanzinformation in einem Unternehmen wird auch als PPR Rückgrat bezeichnet. Automatisierungsstrategien und anderes Unternehmenswissen können im GIS durch Speicherung im IIM zur allgemeinen Nutzung bereitgestellt werden. Anwendungen lesen und interpretieren sie zur Laufzeit.

Die Konzepte des ULEO-Ansatzes wurden für die Produktentwicklung eines großen Automobilherstellers übernommen. In diesem Rahmen wurde der *ULEO-Server* Prototyp weiterentwickelt und befindet sich seit Anfang des Jahres 2005 im produktiven Einsatz. Die Schlußfolgerungen am Ende dieser Dissertation beziehen auch diesbezügliche Erfahrungen mit ein.

Stichworte, die nicht im Titel erscheinen. Features, globaler Informationsraum, semantische Integration.

1.2 Outline of the Thesis and Hints for the Reader

This section sets out the structure and formatting of this thesis and gives hints for its efficient exploration.

1.2.1 Formatting and Organization

Most sections such as this start with a short paragraph informing the reader about the section's contents. Such paragraphs are printed in italic letters.

☞ Paragraphs formatted like this one designate **important notes**.

✍ This is an **annotation** paragraph that provides hopefully interesting information but is not mandatory for the overall understanding of the text.

☆ This formatting identifies paragraphs dedicated to describing **examples**.

Text formatting. Bold printing designates important keywords within the text to draw attention to the contents at a glance. Often, bold words at the beginning of a paragraph briefly introduce the major topic, while *italic letters* introduce terms. Underlining emphasizes words of special relevance or helps in understanding the meaning of a sentence.

This thesis is **organized** into parts, chapters, sections, and sub-sections. Parts encompass chapters that belong together but are numbered in uninterrupted sequence.

References are given in one of two ways: references to Internet home pages within the World-Wide Web (WWW) are given as URLs within footnotes in the text. References to all other kinds of literature are denoted as *[<authors>, <year>]* and can be found in alphabetical order in the Appendix. This includes references to detailed documentation and/or online retrievable documents on the WWW.

1.2.2 Contents

A concise description of this thesis's *technical structure can be found in the following.*

☞ Readers striving for a quick overview of this research can find this information in Part IV – 9.2, which includes an even more condensed description of technical measures which is explicitly stated in Part IV – 9.2.3. The research hypotheses can be found in Part IV – 9.1.

This first part, the Introduction, tries to give the reader the most relevant information for enhanced readability and understanding of the subsequent parts. Please refer also to the appendix *Technical Background Information*. Following the current outline section is the second introductory chapter briefly discussing current challenges for automotive manufacturers and thereby providing the background for the elaboration of the goals of this research, which is done in Part II of this thesis – these goals are not stated but are instead developed throughout the Part II – Chapter 3 *Requirements for Product Development in the Automotive Industry*. The explicit statement of goals is not given until the succeeding Part II – Chapter 4, which hosts also the section Part II – 4.2 stating the research question.

Part III – *State of the Art* analyzes the currently available range of solutions, potentially helping to meet this work's targets. For this purpose, a *Catalog of Characteristics of IT Solutions for Engineering* is formulated to serve as a guideline. Structured according to this work's targets, the actual survey and discussion will be done after relevant research fields have been identified in the chapter *Overview of the State of the Art*. Part III – Chapter 8 summarizes the surveys and discussions held prior to this point and benchmarks needed solutions with those available, thus briefly motivating what was to be developed during this research.

Part IV – *Concepts and Solutions* provides the answers to these demands in three steps: a brief overview explains the major concepts of the new approach termed the *Universal Linking of Engineering Objects (ULEO)*. Step 2 is a detailed derivation and motivation of the individual solutions suggested. The third step, which will be elaborated in Part IV – Chapter 10 *Detailing Some ULEO Principles*, answers some questions in even more detail, yet need not necessarily be read to understand the major notions. At the beginning of Part IV the hypotheses are stated.

After the concepts have been derived and motivated, Part V – *Demonstration: Implementing and Applying ULEO* suggests corresponding software architectures and reports on implementations.

Part VI – *Conclusions and Recommendations* concentrates on lessons learned. The targets of this work will be benchmarked against what has been achieved, with Part VI – Chapter 17 attempting to give a final assessment. The most relevant work that has been published after this research's initial phase will be looked at in Part VI – Chapter 18 and the consequences for ULEO will be discussed. This thesis is concluded in Part VI – Chapter 19 with recommendations and an outlook of future work.

The Part VII – *Appendices* provides a bibliography and two glossaries of terms and acronyms in a coherent text- and in a list format. Additionally, it provides background information useful for the understanding of the other parts of this document. Two appendix sections follow that are concerned with and set out some technical details referenced within the thesis such as the ULEO XML format.

The Part VIII – *Indexes* includes indexes of the figures, tables, and keywords in this document.

The next and last Part IX rounds off this thesis, offering an afterword and an overview of the research work. performed at the Laboratory of Design, Production and Management.

Chapter 2 Current Challenges for Automotive Manufacturers

This chapter sketches major future challenges arising from severe competition faced by all automotive manufacturers today. Options to meet them by using means of information technology are suggested.

Reacting to the keen competition in the increasingly global automotive market, manufacturers are boosting their efforts to achieve the following targets:

Cost Reduction

OEMs seek to cut costs by reducing intrinsic, invisible, and thus undesired, product variety and by increasing productivity of resources. In effect, product development, manufacturing, and change **processes** have to become more efficient that is, faster and hence shorter, with fewer loops and detours, less redundant work, fewer errors, less manual work (more automation of routine work), and continuous optimization of processes. Less variety in processes simplifies change management and support and brings with it a more uniform IT landscape.

Production **resources** have to become more reliable, allowing maintenance times to be reduced. Development tools have to become more powerful in the sense that they must be able to effectively support these new success factors. Other factors are more computer-based simulation and testing instead of expensive physical prototyping (saves loops), higher motivation of the employees (enjoyable processes reduce down-times), and better qualification of employees.

Generally, the degree of reuse of products, processes, and resources has to rise.

Product developments starting from scratch have to be directed within certain tracks. The same product functions must be fulfilled by the same solutions and components. The direct consequences of a reduced product variety are (1) a commensurate reduced variety in manufacturing and further downstream processes such as quality assurance and (2) an eased change management.

Increase in Product Quality

There is a partial trade-off between influencing costs and quality. However, depending on the steps taken, more effective processes can certainly lead to higher product quality – and higher quality reduces after-sales costs. To improve product quality, product development and manufacturing **processes** have to become more transparent and traceable. The number of errors has to be reduced; the likelihood of detecting errors must be increased. Reduced variety of processes helps to reach these goals. Inspection processes for both the resources (materials, tools, etc.) supplied and product components have to insure the desired level of quality. Less variability in **product** components allows for better optimized standard components.

Higher Model Frequency

If more different products are to be developed within the same given period of time, there is less time left for each individual model. Hence, the same challenges exist as set out for cost reduction. In addition, there is a need for more standard components, for sharing components between the model families, and, as a consequence, for closer cooperation between the people in charge.

Conserving the Identity of Brands

There is also a certain trade-off between prominent brands on the one hand and reduced product variety on the other. The smooth path has to be found by the marketing departments based on market monitoring.

Summary

In order to face the above challenges, automotive manufacturers are heading towards the following targets:

~ Products: higher quality, more reuse and component sharing, less variety.

~ Processes: faster, shorter, fewer loops and detours, less redundancy, fewer errors, better error detectability, less manual work, more automation, more transparency and traceability, closer cooperation, more reuse (e.g., of manufacturing plans), less variety, continuous process optimization.

~ Resources: higher quality (higher product quality and greater reliability), less variety, more powerful development tools, more computer-based simulation and testing, higher motivation and qualification of staff.

The desired continuous process optimization includes five aspects:

~ The clear description and refinement of task definitions and contours, interfaces, and co-action in the context of the overall task.

~ Continuous capturing and reuse of company know-how[*] (including procedural information)

~ Increasing parallelization of processes[†].

~ Growing interweaving of individual tasks as products become more complex (e.g., mechatronic domain).

~ Interweaved cooperation and parallelization of processes depend vitally on the informational integration of the process participants (people and applications).

Footnotes

[*] General information

[†] See *multi-modeling* in the catalog of requirements.

Part II – Goals and Motivation

During this part of the thesis, the objectives of this research will be derived and motivated step-wise.

Chapter 3 Requirements for Product Development in the Automotive Industry

This section will elaborate the basic motivation for goals of the research work carried out by (a) providing insights into the current information-technological status in automotive product development and (b) taking into account the challenges to be faced by automotive manufacturers (see Part I – Chapter 2) and highlighting their implications for IT for engineering. This chapter will yield a catalog of requirements for future IT solutions derived by comparing (a) with (b).

Problem domain. The intention of this work is to help companies to face current and future challenges in the overall domain of automotive product development using information technology[*].

✍ Readers striving for a quick overview of this work may desire to skip the first parts of this chapter and move directly to section 3.1.5 – *Summary and Generalization*.

3.1 Spotlighting Product Development at a Major Automotive Manufacturer

This section analyzes the current information-technical situation of practical automotive development using selected scenarios in order to identify needs for future improvements, which, in the end, are the focus of this work. IT aspects of the OEM's product development have been investigated by the author during several years of close contact to engineers and management. It is argued here that the lessons learned can be transferred to the automotive industry in general.

3.1.1 Detail Design, Machining Planning, and Mold and Die Tooling for Powertrain Parts

This section characterizes and discusses a typical part of powertrain development that is – apart from car body development – one of the key domains in automotive engineering.

Multi-Modeling Technique and Vertical Associativity between CAD Models

This scenario portrays the deficiencies of existing uni-directional associativity between CAD models.

The development of powertrain components is strongly characterized by a so-called ***multi-modeling technique***, according to which the CAD model of a product part such as a cylinder head is split up into multiple partial models that are partly related to each other in a certain order. Roughly speaking, a cylinder head is typically split up into a minimum of four models: (1) a *rough-part model* that is used for designing the die casting tools for the rough-part, (2) a *machining model* that is used for planning the machining of the rough-part, (3) a *finish-part model* constituting the state of the cylinder head when ready to be assembled to the crank case, and (4) one or more *base models* containing specific base elements that form the heart of the associativity representation in multi-modeling.

The motivation for this multi-modeling is the desire to enable engineers to do more tasks in parallel: after the engine designer has set up the rough outline of the cylinder head, the mold and die tool designer (MDT designer) is able to start working on the die. After the next step of detailing by the

Footnotes

[*] IT for Engineering

engine designer, the MDT designers may continue their work and so on. Between these processes are joint sessions with both engineers.

The **associativity** between the models is facilitated by the CAD system deployed, viz. CATIA^{TM} Version 5, by means of uni-directional links, the so-called ***multi-model links (MMLs)***. Such links are maintained between a geometrical object (in CATIA, called a feature) and any of its copies. The latter are created by a special command and can be further modified after their creation, for example, by adding a thickness. The original feature, e.g., a spark plug hole, is stored inside a base model while its copies can be placed into any of the other models (see Figure 1). If the base feature changes its location, orientation, or extension, the correlated copies can be updated by loading the respective CAD models and clicking an update button. If the copies are moved, the base feature is not affected – this is the effect of the uni-directional link. This uni-directionality of the MMLs is the ultimate reason for managing base models. If several CAD models (also termed *MML models*) have to be associative amongst each another, utilization of such base models suggests itself.

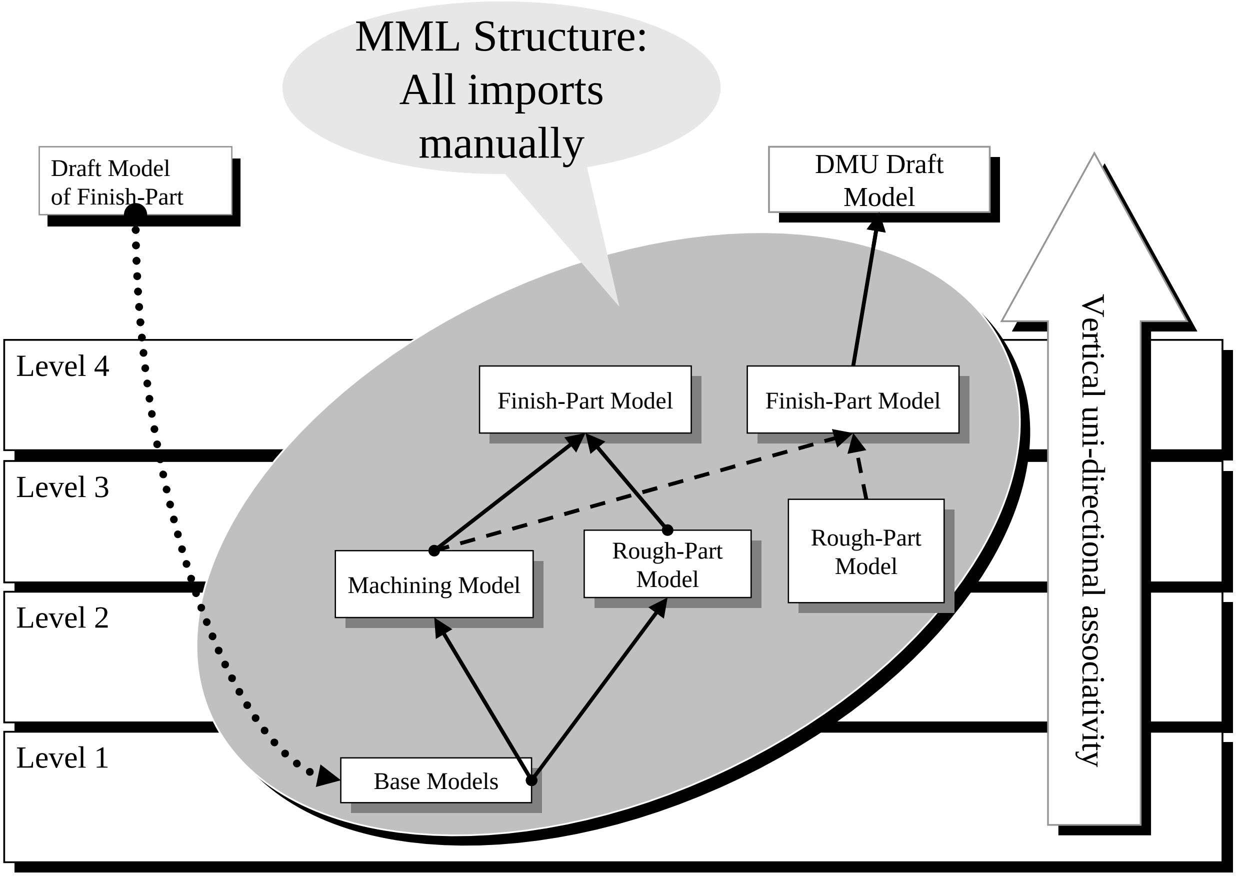


Figure 1: Multi-Modeling

The finish-part model is the result of a CAD operation: the logical subtraction of the machining model from the rough-part model yields the finish-part. The finish-part does not contain any objects other than the overall finish-part. Features, for example, are only found in the logically subordinated models (rough-part, machining) and not transferred into the finish-part model.

As no links lead from the finish-part model to the other models[*], engine designers have to work indirectly: they manipulate the base model and the machining model if they want to insert a spark plug hole. However, designers would rather work with the finish-part model as it is commonly not intuitive for them to work with rough-part and machining models. **Bi-directional associativity** would be one pre-requisite to allow them to do so.

Of course, also **changes** executed inside a certain model cannot be propagated downwards within the MML hierarchy and thus have to be carried out within the lower models and propagated upwards in order to obtain the results. For example, changes to cylinder heads often have to be verified by considering the detailed geometry inside the finish-part model before being released. Today, the designer has to create temporary solutions inside the finish-part model and – if successful – reconstruct them, starting from the MML base level models, and propagate them upwards through the model hierarchy. This lack of intuitivity reduces the **efficiency** of the work, as these kinds of changes are very time-consuming and, on the other hand, occur frequently. As a consequence, a remarkable saving of time could be achieved if a faster method would exist.

✰ This will be demonstrated by a realistic **example** of the current sub-optimal situation. Assuming that the car engine designer wants to create an additional crank case ventilation hole, the following steps are required:

(1) The designer creates test bodies inside the experimental model. One of them will become part of the core of the oil jacket.
(2) Distances to other features are short. The designer needs a realistic geometric model to check the minimum wall thicknesses. Thus, the casting-specific drafts have to be considered.
(3) The designer copies the test body into the real MML model of the oil core to check the compatibility of contours.
(4) The finish-part model is updated to check the contour on the finish-part (subtraction of the modified oil core from the finish-part).

If the changes implemented do not result in errors or collisions, the experimental results have to be taken over into the MML structure. This means that the following steps are required:

(5) The designer reduces the body back to a primitive and copies the body from the draft model of the finish-part to the corresponding base model for contour elements.
(6) The experimental elements are removed from the finish-part model and the oil core model.
(7) The designer then imports the base body into the upper oil core model.
(8) The detailing has to be done again (drafts, fillets, mold joints).
(9) Finally, the designer updates all the affected models in the hierarchy.

Footnotes

[*] All uni-directional links lead to the finish-part model.

Downstream Process Mold and Die Tooling and Horizontal Associativity

This scenario demonstrates the need for horizontal associativity between models of different steps in product development.

Before starting with the work, the mold and die tool designer creates a copy of the master rough-part model. As illustrated in Figure 2, today, the information flow is largely uni-directional and CAx-based or without any CAx support.

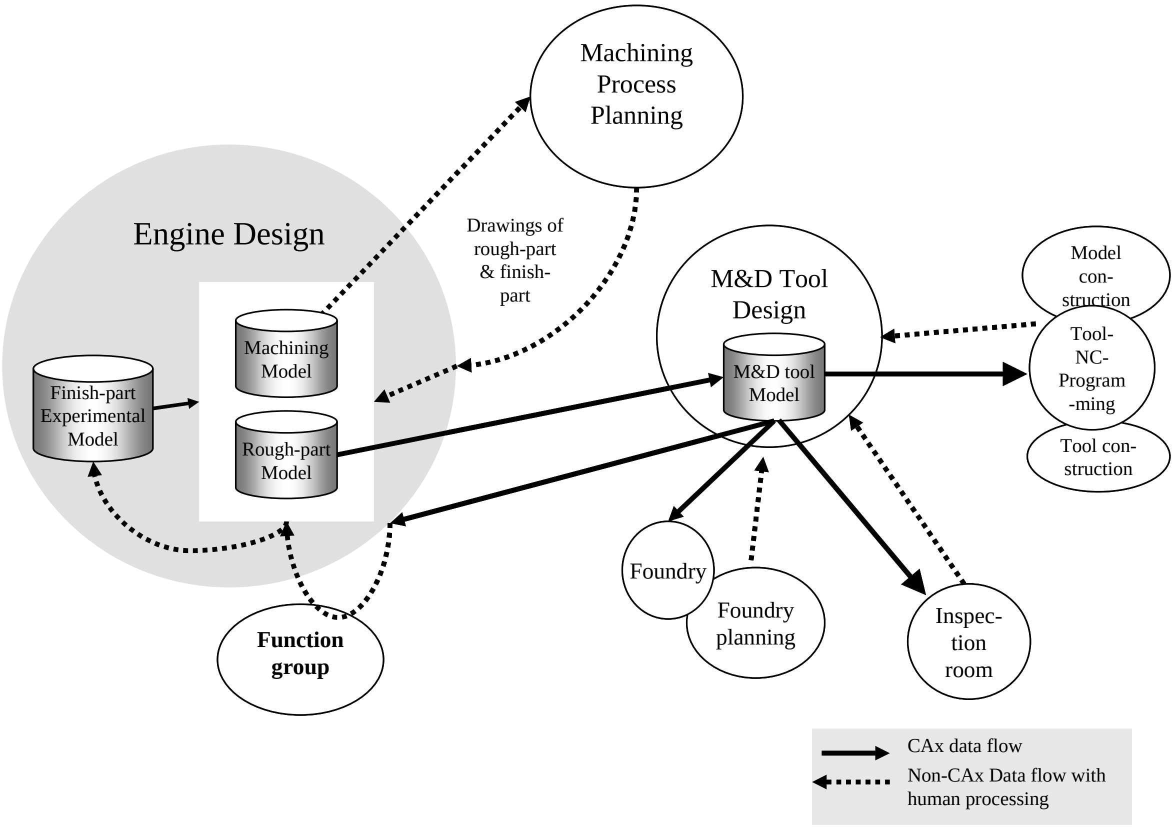


Figure 2: Cylinder Head Design in Context

During the course of the work, modifications of the rough-part are typically called for: the MDT designer imports objects from the rough-part model into the MDT models, where they usually have to be adjusted in some way. These changes must be tracked inside the master rough-part model, but they cannot automatically be propagated back to it. Hence, the resulting MDT model is copied back to a central exchange medium (a so-called exchange map) where the designer can access it and manually track the MDT designer's changes inside the master rough-part model.

☞ This is another case of uni-directional associativity. As indicated by Figure 3, it could be termed **horizontal** in contrast to the vertical uni-directional associativity between multi-models.

Discussion of Possible Improvements Regarding Associativity

The section focuses on the benefits of a future bi-directionality between CAD models.

As has become clear from the preceding two sections, bi-directional associativity is very desirable (at least) in this domain. If both horizontal and vertical bi-directional associativity are available, the term ***multi-directional associativity*** is applied. Figure 3 gives a simple illustration of this concept.

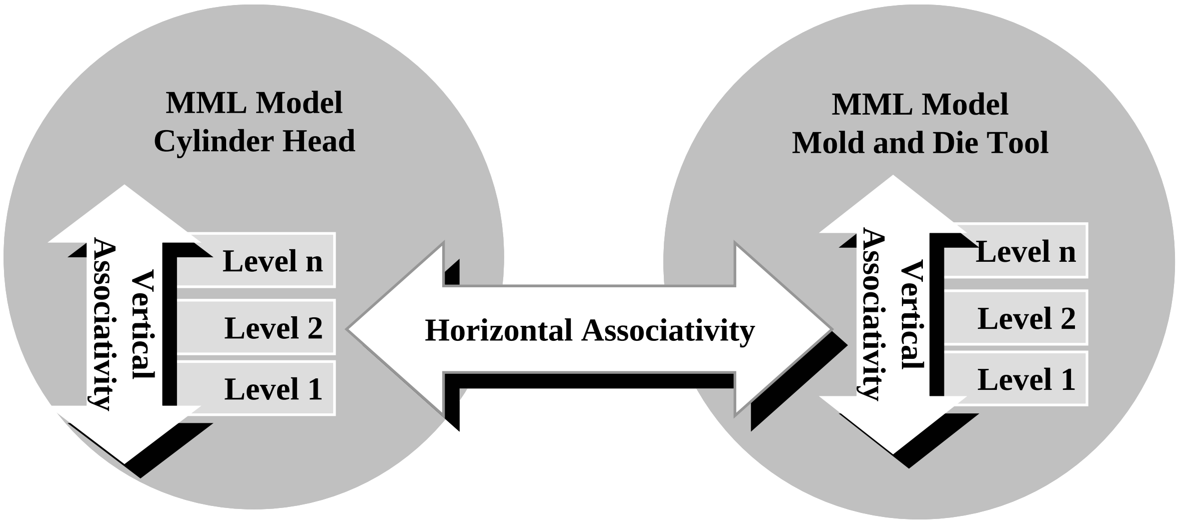


Figure 3: Multi-Directional Associativity

Some **benefits** of multi-directional associativity are set out below:

(1) Cylinder head and MDT designers can execute changes on the part from within any MML model, and changes are propagated in the form of change requests to the related models (also top-down and right-to-left). “Change request” in this context means a message to the engineer who is responsible for a certain sub-model, stating a running change activity. The respective features are highlighted in the CAD systems. The recipient accepts or denies the proposal or makes a counter-proposal.

(2) Designers can work simultaneously on the same finish-part under the following prerequisites:

(2a) There is a means of propagating the changes from the high-level finish-part model to the lower-level rough-part and machining models. Top-down associativity additionally to bottom-up leads to bi-directional associativity.

(2b) There is a way to synchronize the actions of several designers working on a set of MML models that describe the same part (need for workflow assistance). The support by a (simple) workflow management system could pay off in this context.

(3) The drawbacks pointed out in the preceding examples are avoided:

(3a) Non-intuitive and intricate work becomes more purposeful. Designers can perform their tasks straightforwardly. For example, changes to the finish-part can be made directly in the finish-part model itself.

(3b) Current time-consuming deviations in engineering work caused by the need for consecutive one-way associations are eliminated.

(3c) Computer-aided change requests significantly reduce the probability of errors arising from the currently orally conducted exchange of information between the participating engineers.

Downstream Process Machining Process Planning

This section targets the shortcomings in machining process planning that result from uni-directional associativity.

The overall cooperation between engine design and machining process planning shows characteristics similar to those cited between engine design and mold and die tooling. Since the designers employed by the automotive OEM (being the subject of the investigations discussed in this thesis), work with a set of user-defined features to construct the machining model*, the task of machining planning can be additionally supported by software†: the machining planning engineer‡ selects certain features inside the machining model and the system suggests appropriate machining operations. However, since the resulting associations between features and machining operations are also merely uni-directional, changes that the machining planner finds mandatory, have to be routed back via telephone and tracked inside the master models by the engine designer.

The benefits of bi-directional associativity would, in principle, be equivalent to the ones noted in MD tooling.

Discussion of the Introduction of Finish-Part Features

Dedicated and optimized logical building blocks for product models can be provided through finish-part features. This section gives some details.

Every software application along the product development process chain (ProSAp) has to fulfill specific tasks using specialized technical knowledge and modus operandi for problem-solving. As a result, each ProSAp has its own **view** of the *products, processes, and/or resources* (***PPR***). And managing these views is not equivalent to managing multiple views of a single master product or PPR model. IT solutions must adapt to these different characteristics of development tasks if they want to be able to represent effective and intuitive tools and aids for the engineers. Feature types, for example, should correspond to the concepts the engineers have in mind when performing their tasks. **Features**§ take on the role of building blocks for creating view-specific product and PPR models.

In the current scenario, finish-part features would promote intuitive working directly on the finish-part model. Moreover, the lack of features within the finish-part model can be seen as a principle drawback affecting most downstream processes negatively, as they depend on objects that they can relate their own models to. If this is not possible, downstream models are more or less cut off.

☆ Regarding **quality assurance**, for example, the following deficiencies must be pointed out:

- ~ Since there are no finish-part features, engine designers have no means of entering tolerance information into the finish-part model that is attached to functional objects.
- ~ Furthermore, the quality assurance process cannot retrieve design features from the finish-part model for inspection planning.
- ~ Product changes in terms of modified, inserted, or deleted features cannot be recognized by software.

Footnotes

[*] So-called *PowerFeatures*

[†] These features have to be instantiated by the engine designer into the base models and copied with link to the machining and other relevant models.

[‡] Using another so-called workbench of CATIA™ V5

[§] Amongst others; see below

While domain-specific feature types have a variety of benefits, there is also a downside: their usage may lead to a **logical separation** of the feature-based models produced. As Salomons pointed out in *[Salomons, 1995]*, this ***multiple view problem (MVP)*** originates from object orientation, but also affects features (and other EOs) in engineering. This has to be tackled by the informational integration of applications. Moreover, in order to avoid confusion about feature types with the same name but defined in different development steps (domains), the respective **context** has to be explicitly specified for each feature type as well as for any other type of informational entity. This also implies that the corresponding **meaning** of such entities is context-dependent, as well.

Summary of the Scenario and Conclusions

The benefit of the MML methodology is the facilitation of semi-parallel work: it is achieved by relating features uni-directionally within different MML models. However, the author has stated that MML methodology and its uni-directional associativity also yield several significant **drawbacks**:

- The lack of features in the finish-part model, which leads to cut-off downstream processes.
- Engine designers may not work inside the finish-part model although this would be intuitive for them.
- The associativity granted by this method is not only uni-directional but also purely geometry-centered. This prohibits the representation of other kinds of relationships between features.

The availability of **multi-directional information flow** as well as **associativity** and **finish-part features** are strongly desired by the designers interviewed by the author, since they could avoid the deficiencies cited above. In addition, they suggested that associativity between features should be deactivated if desired (for features that are used merely for auxiliary purposes).

Semantics of informational entities is context-dependent by definition, which means that each IE is only valid within a specified context. The representation has to adapt to this fact.

3.1.2 Quality Assurance for Body-in-White Parts

Also standing for other car body development processes, the quality assurance process for body-in-white parts is outlined within the current section. As with the scenarios discussed in the preceding sections, the situation at a major automotive manufacturer is elaborated.

Overview of the Quality Assurance Process

This section shows first steps from a very heterogeneous software landscape to an integrated one. However, to achieve significant improvement, a new approach must be available. Interested readers can find more details and a history of IT for quality assurance in [Zimmermann et al., 2004].

Today, the quality assurance process investigated is mainly characterized by a rather inhomogeneous IT landscape combined with a certain amount of paper-based work. The CAD system CATIA^{TM} is employed for designing the finish-parts[*] (multi-modeling as applied in the powertrain domain is not utilized here). Various steps have been and are still being taken to make this process more uniform and to reduce manual and paper-based work for supporting downstream processes:

Footnotes

[*] The notion of finish-parts will be used here for single components as well as for assemblies.

- ***Car body features*** have been introduced for finish-part design. They help to reduce product variety and accelerate the design process, especially change processes. As discussed in the previous section, they are the basis for linking downstream processes to the design phase.
- **Scripts** (also termed macros) can be fired inside CATIA to support the manual insertion of simple features for inspection planning into copies of the finish-part model and to export these inspection plans as comma-separated csv files.
- Several **systems** are used for dedicated purposes in inspection planning and programming, depending on their individual characteristics. And a first bridge has been built between one of them – SILMA's offline inspection programming system CIMstation^{TM} – and the finish-part design in CATIA^{TM} (see *[Petkova, 2001]* for more details): the inspection programmer may import the above-mentioned csv files containing the inspection plan of a certain part or assembly. A script inside CIMstation^{TM} recognizes the types of features inside it and visualizes them, together with the associated part geometry. Now, the engineer may select the features to inspect and manually specify the device path for the coordinate measuring machine (CMM). Afterwards, the script generates – type-dependent – a piece of DMIS code and writes it to another text file (DMIS file).

☞ What happens here is the automatic application of hard-coded **inspection strategies** for generation of inspection programs. Points and circles inside the inspection plan are transformed into points serving as measuring elements (points that a tactile measuring machine is to hit). Inspection strategies, like all other strategies, make up an important part of the company's know-how.

- Additionally to CIMstation^{TM}, an inspection programming system called DELMIA inspect^{TM} promises to be a usable candidate, but has not yet reached the desired state. When fully mature, it is expected to provide Dassault family-typical uni-directional geometry data flow and uni-directional associativity. Inspection strategies will be available in a very restricted version, with parameterized, built-in inspection features storing them; user-defined features will not be supported in this sense.

Where the CIMstation software is not used, inspection strategies are stored inside the experts' brains and the same tasks have to be performed manually. The DMIS file containing a complete inspection program is converted by a post processor into BNC[*] code and run on a coordinate measuring machine. The **inspection results**, also called *actual values*, are stored as sets of records within databases. There are no links back (that is, no associativity) to the tolerances and features.

Analyses of the actual values are designed manually without any links to the inspection strategies employed. The actual value database systems frequently offer this functionality.

Discussion

The offline inspection process described still leaves much room for improvement:

- The **information flow** between applications is largely restricted to plain geometry not containing any functional objects and semantic information. So, work has to begin almost from the scratch at each process step. And while the CIMstation chain brought a significant improvement, it is only usable in a small part of the company's quality assurance. Also with CIMstation, only a uni-directional information flow is possible. Throughout the CAD/CAQ process chain, there is no associativity between product features whatsoever. In the future, the uni-directional associativity of DELMIA inspect^{TM} features promises to allow product changes to be tracked better than today. Still, there will be no way to track back information gained from analyzing inspection results.

Footnotes

[*] Binary numeric control

- Inspection-, analysis-, and other **strategies** are for the most part undocumented and non-standardized. CIMstation^{TM} is a step in improvement, but here the strategies are coded in a proprietary format and there is still no computer-aided management for them. DELMIA inspect's^{TM} strategies are also proprietary and not universal enough to handle new kinds of inspection situations such as flanges and compound features. This is a significant shortcoming, as standardized and flexible strategies are of great benefit, since they are a presupposition for **comparable** inspection processes and results, and thus comparable propositions about product quality, within the same production plant and also between plants.
- There are still many manual steps, most of which are routine work and error-prone. Since inspection programming can be automated to a large degree, **automation** could help to ameliorate this situation.
- Inspection plans are stored in files, without any IT-supported associativity to the CAD system or to the more downstream inspection steps. **Product changes** thus cannot be tracked and have to be coped with manually.

Inspection-plusplus Workgroup

This section brings up an issue that actually belongs to the implementation section of this thesis. However, to reflect a realistic state of affairs in quality assurance at the time this thesis was finished, it is briefly mentioned here, as well.

✍ The ***Inspection-plusplus (I++) Workgroup***[*] (say: »I plus plus«) is a workgroup of German and Swedish automotive OEMs[†] and inspection equipment manufacturers. It consists of experts in the fields of product quality assurance and information technology and is working on commitments that are to enable or push the interoperability of software applications along the CAD/CAQ process chain. The author takes part in this effort. The workgroup's ultimate motivation is to obtain or offer, respectively, a variety of compatible software tools for quality assurance (see *[Zimmermann et al, 2004]*).

By the end of the year 2004, substantial work was done in developing a common **information model** as a basis for the exchange of information (see also section 13.3.1). Figure 4 illustrates the *Quality Criterion* relation in its closer context.

Footnotes

[*] See the URL *http://www.Inspection-plusplus.org*

[†] Such as BMW AG, DaimlerChrysler AG, Volkswagen/Audi AG, and Volvo AG

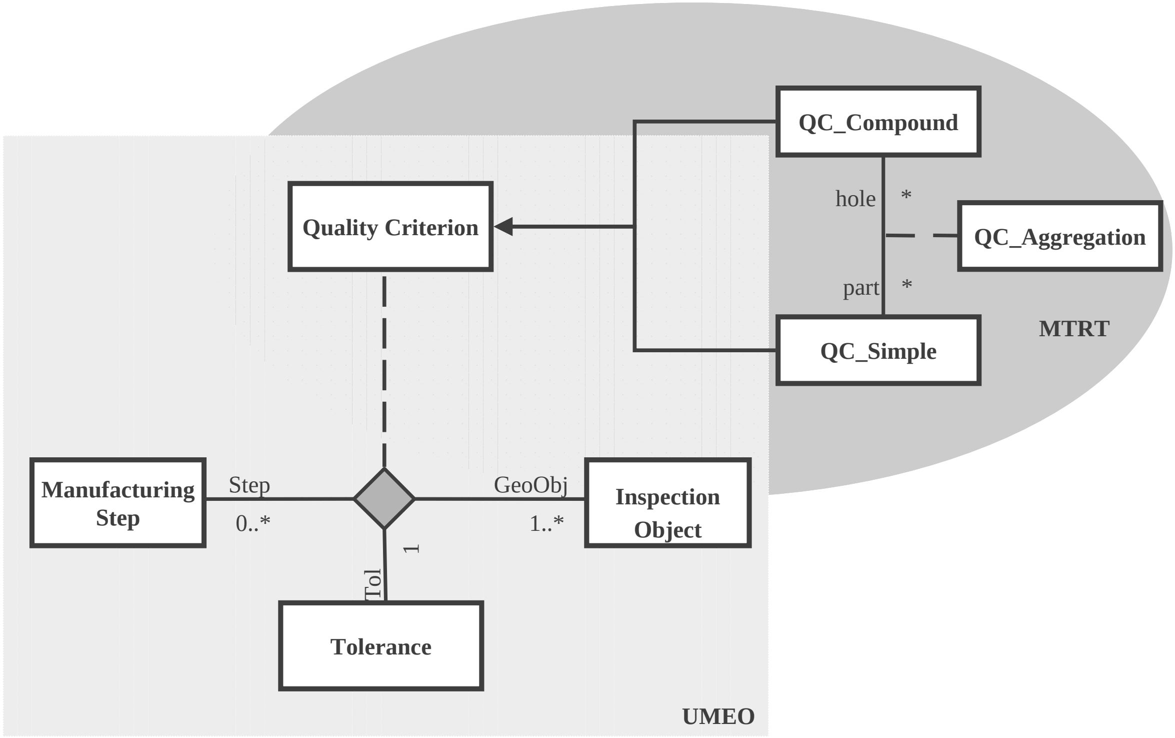


Figure 4: Impression of the I++ Information Model: The Quality Criterion Relation

Summary and Conclusions

In the quality assurance domain, severe shortcomings that need to be remedied are evident:

~ The need for integrated process steps and applications has already been recognized, as the I++ efforts indicate. **Integration** as used here means facilitating bi-/multi-directional information flow and associativity between the respective informational elements such as features or other engineering objects. **Relations** between informational elements are the technical foundation for achieving associativity. They must not be purely geometry-centered, as there are many more kinds of relevant relationships to be considered and deployed. For example, inspection strategies can be intuitively viewed as relations between several objects, as they represent the link between two process steps in quality assurance and are, thus, paramount for their integration. They are the starting point for the inspection planning process. The CAD/CAQ information model developed by the I++ working group (see also the section *Inspection-plusplus*) underpins this and pinpoints many more examples for this issue, emphasizing the necessity for a **sophisticated representation** of relations. The overall product information, which consists of the systems' respective individual contributions, must be **organized** and **structured clearly**, allowing users and systems to access information quickly, thus avoiding redundancy.

~ The degree of **automation** has to be increased. This can be reached most effectively by tackling inspection strategies. **Strategies** of different kinds have to be stored outside of any program code and managed and automatically interpreted by the software systems. As several systems need to access these strategies, they have to be represented in a **universal language** that is flexible and powerful enough to meet future requirements (see above).

3.1.3 FEM Simulation for Car Bodies

This section gives an introduction of the CAE process chain, where similar deficiencies are found as worked out in the previous scenarios.

Situation

CAE is a purely computer-based technology that aids in developing a product to a high degree of maturity before first physical prototypes are employed in real experiments. Computer-based stiffness calculations and crash simulations are examples for typical CAE tasks. In effect, the number of very expensive crash tests is reduced.

Also in the CAE domain, various **software systems** are used, mainly because of their individual benefits. Although, for example, the CAx system CATIA^{TM} also includes modules for finite element meshing and solving, many experts prefer other vendors' solutions such as the MEDINA^{TM} mesher and Nastran^{TM} or LS-Dyna3D^{TM} solvers. As CATIA^{TM} is the strategic CAx system deployed at the OEM investigated, it is necessary to transfer data from the CAD system into the mesher software.

So far, pure geometry information is transferred using standard exchange formats. The **present approach** is to additionally transfer feature information. As already explained, dedicated car body features are available and in use. Utilizing this information in the creation of the mesh allows adaptation of the mesh parameters to the product's needs. This means that areas that are of special interest because of their relevance for stiffness[*] can be meshed more densely while other areas can be meshed wider. In the future, the features themselves could also carry mesh parameters and other meshing-relevant data or could be associated with them. This leads to a mesh that is – although created automatically – to a great degree optimized and solvable within a reasonable period of time[†].

Hence, the **challenge** is to transfer feature information from the CAD system to the mesher. In a newly implemented scenario, geometry information is exported from CATIA^{TM} in VDAFS[‡] format. VDAFS is a text-based format with the option of carrying comments. This characteristic is employed to represent feature information, in particular, about the geometric faces belonging to an individual feature.

On the mesher's side, an import script can be fired, collecting the feature-based information and detecting the feature type to apply a hard-coded meshing strategy in order to automatically produce a high-quality net.

Discussion

There is a **uni-directional information flow** from the CAD system to the mesher, yet there is **no associativity**. The pros and cons of this situation have been discussed earlier.

Data exchange using legacy **neutral exchange formats** raises a problem that has not yet been discussed. As in this example, transferring information this way can **never** capture **all** the **information** inside the CAx systems, as their vendors always try to include special proprietary functionality (and corresponding data structures) as unique selling propositions to stand out in the pool of competitors and the exchange formats are not flexible enough to cope with this factor. In addition, file-based data exchange generates **redundancy** in principle and is not as efficient as delivering

Footnotes

[*] Such as beads

[†] Today this is still a major concern. Solving of a complete car body takes time periods up to several days.

[‡] VDAFS files are ASCII files that describe 3D CAD models; see the glossary for *Vereinung Deutsche Automobilindustrie Flächen Schnittstelle*.

information at the time needed (through the online availability of information). As in the case of the current example, only a very small portion of the **semantics** of the transferred information is passed on, making usage of a highly-specialized kind of hard-coded solution with comment lines necessary. In turn, semantically more powerful representation formats allow for more flexible software solutions as they are able to recognize and react to the types of imported objects. They should be as self-explanatory as possible.

The current CAD/CAE process chain employs **hard-coded strategies**. As argued above, new design feature types will call for new strategies, therefore requiring modification of the code. Additionally, in this domain, several FEM solvers might be used, necessitating that the strategies be represented in a neutral format and in a neutral location.

Summary and Conclusions

The IT situation in the FEM domain resembles the scenarios presented in the preceding sections:

- A **multi-directional information flow** would allow information to be routed back to the designer.
- **Associativity** would make change processes much more robust.
- Handling of **strategies** leads to the same conclusions drawn above.
- Transferring the (context-dependent) **semantics** of transferred information is an important prerequisite for achieving flexible software applications able to cope with future types of objects such as features. Of course, this holds true in general and not just for the FEM domain.
- In general, an information flow based on neutral **representation formats** should not be tailored to certain sub-domains but based on flexible languages instead. Flexibility in this sense can be achieved be offering high expressiveness[*] combined with the ability to transport context-oriented semantics (see previous point). Thus, the language should be universally applicable in order to represent a large variety of information, while describing enough of the meaning of the represented information to enable information users to correctly judge and utilize it.
- An **online information flow** between applications helps to avoid redundant information storage and allows faster and more targeted information access.

3.1.4 Car Body Product Data Management

Looking at the domain of car body engineering, this section discusses some general problems related to the inter-application flow of information.

Situation

Information about parts and (sub-)assemblies and their positions within the respective super-assemblies are created using the CAD system. In addition to this purely geometry-oriented information, clamping-, tolerancing-, welding-, multi-model link-, and other information is associatively[†] stored in these models. Although very relevant for downstream processes, this proprietarily stored information cannot be fully accessed by downstream processes. The reason for this is the software manufacturers' policy of disclosing the respective APIs restrictively, probably motivated by the wish to sell their own PDM systems for handling and utilizing such information.

Footnotes

[*] where needed; unnecessary expressiveness is to be avoided.

[†] uni-directional

Discussion

In order to solve the problem, the automotive company can either buy the software vendor’s PDM system or **re-implement** certain portions of the CAD system’s **functionality** using the respective APIs in order to manage the corresponding data by itself.

While, as a matter of fact, costs for software implementation and maintenance are always critical for the practical application of new concepts, the negative effects of being restricted to the software’s functionality should not be underestimated. The deficits in handling **associativity** have already been discussed and must be judged a major drawback, as will be argued later. The same holds true for the handling of **company knowledge**, e.g., in the so-called *Knowledgeware*^{TM}, which is a rather rudimentary knowledge-based add-on to the CAx system CATIA^{TM}. Hence, while the single-CAx platform strategy followed here does not help to make the software vendor more flexible, it rather deteriorates options for considering new concepts in productive product development, including the openness of information to any other vendor’s software. The alternative solution of sharing information within a vendor-neutral, **global information space** becomes more attractive. In spite of the restrictions* mentioned in accessing the information via APIs, the problems faced by current downstream applications strongly support the same approach.

Furthermore, it becomes obvious that a PDM system can support the product development processes more effectively if it “understands” the **semantics** of the information it manages. For example, if multi-model links are known to the PDM system, it may identify CAD models affected by the changes in another CAD model and have them updated without having to check them out and in again. This is, however, only true for a more **intelligent kind of PDM system**. There is also another philosophy conceivable, where PDM systems are reduced to their very task of storing information, while the intelligence is kept outside.

Summary and Conclusions

In order to achieve significant progress in today’s product development processes, it has been argued that the software vendor-independent approach of sharing information between applications, including downstream applications within a **global information space** is the best concept.

It has also become clear that it is important to handle and transport (always context-dependently) the **semantics** of information and that “intelligent” versions of PDM systems take this into account.

The **availability of software APIs** to access the proprietary information and make it available for others has been pointed out as being a possible critical **bottleneck**: it is hard to facilitate information flow on a neutral basis, if certain software vendors prevent information from being read by other applications. **Workarounds**, however, are available: one alternative would be to convince the software manufacturer to change this situation; another is to replace parts of the commercial software’s functionality and respective information management by an open solution (partial re-implementation) while keeping those parts of the proprietary software and data management that are open for access. Or, and also in cases where APIs are not available, there is frequently a means of influencing an application’s behavior through import/export files.

Footnotes

* It has already been argued that a lack of expressiveness of existing neutral representation formats can be counteracted by formats offering optimized expressiveness and the ability to transport semantics.

3.1.5 Summary and Generalization

After a summary of the preceding discussion of automotive product development, the generalization of the results to other software systems and automotive manufacturers will be motivated.

Summary of Conclusions from Practical Experience and Surveys

In the ending section 3.1, spotlights have been thrown on the current state of information technology in automotive product development. Although these examples do not provide a full overview, they are presumably typical for the overall situation in this field. References from literature and the Internet such as the NIST *Product Engineering program*[*] confirm this impression.

The investigations revealed significant potentials for improving the current product development practice. Today's situation is mainly characterized by (1) a highly heterogeneous software landscape (motivated by the philosophy of using best-of-class solutions), (2) a dramatic lack of information sharing and reuse between the software applications, and (3) a significant amount of automatable, error-prone manual routine work. Finally, (4) currently applied solutions for achieving parallel work yield significant drawbacks. Reasons for this situation are manifold. Some of the most prominent ones are certainly organizational ones: traditionally, automotive development has been carried out sequentially, clustered into the individual engineering domains. The companies' organizational structures, and most importantly the profit center approach, amplify the persistency of this situation and hamper joint efforts for the integration of processes and IT tools. Following the dictum of supply and demand, the available software products mirror these facts. Although individual vendors try to offer integrated product families for covering several CAx fields, they must still be judged to be not exclusively composed of best-of-class components but also to offer an integration that is far away from the state of the art (compared to Semantic Web approaches and others) as they stick to highly specialized relations and associativity – most often geometry-oriented. All the available automation solutions encountered during this research such as dedicated solutions for mold and die and fastener design, fall into this category. To a certain extent, this seems surprising, as the software solutions used in-company might be applicable for more sophisticated concepts, as the validation of available APIs hints. It thus seems to be a more philosophical and conceptual problem, again ultimately dictated by supply and demand. Unfortunately, science also seems to follow different targets, as the survey of the state of the art will show (see Part III – Chapter 8 for a summary).

More exactly, the following **key conclusions** have been drawn from an IT-biased perspective:

- Generally speaking, uni-directional **associativity** is of importance for making change processes efficient and robust. Multi-directional associativity is necessary for a backward flow of information, the importance of which has been shown. It should be possible to deactivate associativity under certain conditions. Associativity is achieved by creating and managing **relations** between pieces of information, also called engineering objects in this thesis, and by providing software algorithms that utilize them. Hence, the provision and usage of relations are significant and core means for integration applications.
- **Parallel work** in engineering can be supported by multi-modeling techniques. Current drawbacks are severe, however. They can be remedied by introducing multi-directional associativity and finish-part features.
- Managing the **semantics** of information is important for flexible software applications and more powerful PDM systems. As all information is context-dependent by nature, this is also true for

Footnotes

[*] See the URL *http://www.mel.nist.gov/proj/pe.htm.*

information on semantics. Consequently, it is also necessary to be able to represent and manage **contexts**.

- Software vendor-independent information sharing is a prerequisite for introducing significantly more powerful IT concepts. Information must be shared between software systems inside a **global information space (GIS**). The bottleneck for existing applications, i.e., availability of software APIs is cumbersome, but workarounds exist.
- A clear and structured **organization of the information** inside the global information space is essential.
- Neutral representation formats for the information exchange and sharing between applications must be highly expressive and able to transport semantics. Desired languages are universally applicable, describing a sufficient part of the semantics of the information represented. Within a global information space, the use of a single, **primary representation formalism** as the basis of a common communication infrastructure is suggested. Other languages can be transformed to and from it. The contrasting approach of using multiple primary formalisms without a central base formalism is rejected due to the high costs and low flexibility of n-to-m translators.
- An **online information flow** helps to avoid redundant information storage and facilitates faster and more targeted information access. This completes the idea of a global information space existing between multiple running applications.
- **More automation** is needed, especially through automatically interpreted strategies. Formalisms for their representation should be open, flexible, powerful, and outside any program code. Strategies should be accessible through the GIS. Application-spanning strategies favor the option to handle **control information** within the GIS, as well. It has been shown that strategies link input engineering objects to output engineering objects and are therefore naturally representable within **relations** on the abstract (class) level. Again, relation handling is crucial for integrating applications.

Generalization of the Conclusions to Other Automotive Manufacturers And Other CAx Systems

The product development conditions at the automotive company studied in this thesis, including the state of the information technology deployed are assumed to be very similar to those of other automotive OEMs. This assumption is based on the following indications:

- A large majority of automotive manufacturers also uses the CAx system CATIA^{TM} from Dassault Systèmes.
- Publications for scientific conferences and workshops, and discussions with participants confirm this assumption (see the NIST *Product Engineering program*[*], *[Zimmermann et al, 2002a+b]*, and *[Zimmermann et al., 2004]*).
- Discussions at Dassault Systèmes user conferences confirm this assumption.
- Multi-vendor working groups and projects, e.g., I++, GACI (German Automotive CATIA Initiative) confirm this assumption.
- The international patent situation points in this direction.
- Information from DaimlerChrysler AG management underpins this assumption.

Footnotes

[*] See the URL http://www.mel.nist.gov/proj/pe.htm.

Taking a more in-depth look at the first point, four **key software packages** lead the field:

(1) The CAx system **CATIA**^{TM} from Dassault Systèmes is used by (at least) the following companies for (at least) car body development:

~ Audi AG (outer skin), BMW AG, DaimlerChrysler AG, Ferrari AG, all the French manufacturers, Porsche AG, and Volkswagen AG (outer skin).

~ Toyota Motor Corporation is currently in the process of replacing a proprietary system by CATIA^{TM}.

(2) **Pro/Engineer^{TM}** from Parametric Technology Corporation (PTC) is used by Audi AG, BMW AG, and Volkswagen AG for powertrain development.
(3) **Ideas^{TM}** is used by Ford AG and Volvo AG.
(4) **Unigraphics^{TM}** from Electronic Data Systems Corporation (EDS), is used by General Motors Corporation including the German Adam Opel AG.

CATIA^{TM} can be characterized as a modern CAx system; nevertheless, the other CAx software products mentioned offer comparable functionality not deviating in a quality or quantity significant from the viewpoint of this research.

☞ Thus, the conclusions formulated in the preceding section for specific CAx systems and for a specific automotive OEM can be considered generally valid for all major CAx systems and all major automotive OEMs. The same assumption is made for the discussions and conclusions in the next section 3.2. Therefore, the goals of this work are considered to be of general interest in the automotive industry. As will be shown in the survey of the state of the art, the respective scientific communities also do not offer an integrated or compatible set of solutions for matching this research’s combination of goals. For this reason, this research is also considered relevant from a scientific viewpoint.

3.2 What is needed? Deriving Requirements for Future Product Development IT Tools

Setting the high-level goals targeted by automotive manufacturers (see section Part I – Chapter 2) against the results of the section Summary of Conclusions from Practical Experience and Surveys, *the section at hand identifies actions that will support reaching such targets. This yields a catalog of the requirements that future IT solutions for engineering should meet.*

What stands out is that both the issues derived from analyzing the practical situation in product development and the requirements originating from external influences point in the same direction. This means that it is to be expected that companies will achieve great benefits by remedying the shortcomings in the information-technological supply of their product development processes.

✍ The following quotation from Vliet and Luttervelt, who also give an overview of design-for-manufacturing approaches, underlines this point (see *[Van Vliet & Van Luttervelt, 1999]*): “*Information handling accounts for more than 90% of all the costs related to human activities in manufacturing. Effective structuring and representation of information is needed to improve the transformation of information and the controlling of the design and production process. To be able to deal with the complexity involved in information transformation, it is needed to use all relations that connect the information carrying elements [Kals & Lutters, 1998].*”

The following sections discuss the aspects of this issue in more detail. They are organized in line with the key requirements set out in the preceding sections, namely the informational integration of applications, process automation, and, as a further answer to the requested powerful development tools, usability. A discussion of the prerequisites for the practical introduction and applicability of new IT concepts complements the set of issues in this current section.

3.2.1 Informational Integration of Applications

As set out in the definition of this notion in the appendix *Coherent Glossary of Important Terms*, the informational integration of applications ideally means – from an information-technological viewpoint – facilitating a high-level information flow and sharing between applications, multi-directional associativity, and transport of context-dependent semantics within a continuously existing **global information space (GIS)**. The information may consist of declarative and procedural components and of specific and general information. This definition implies the presence of some form of inter-process communication between the applications in the GIS.

✍ Integration is desired in almost any situation where more than one software application is used to support a process, contributing to reach a common goal. The common goal is the ultimate source for the need for integration of the process steps (tasks), i.e., to synchronize sub-processes and to provide the output of certain sub-processes as input for subsequent sub-processes. In the case of software applications, input and output consist of information. To maintain the output/input flow, output information must be understandable for the sub-processes (software applications) consuming it. One option to achieve both synchronization and input/output flow is to let humans read the output information and enter it manually into subsequent software systems at the right time, thus transforming the information while performing a semantic interpretation and mapping of the two systems' informational entities. While this procedure generally works, it shows some significant and obvious drawbacks: manual interpretation and translation are subject to errors and are – compared to automated solutions – slow. Moreover, the knowledge used to bridge the applications is not explicitly documented but hidden in the human beings' heads. Also, the time periods between the start of two applications might be unnecessarily long due to the humans' occupation with other tasks. These considerations yield the **basic targets of informational integration** approaches: to correctly translate output to input information and to route it from a sending to a receiving application. On top of such informational integration, synchronization of applications can be tackled.

Global Information Space

Facilitating a **high-level flow of information** between all people and software systems, while each informational entity is accessible to anybody at any time (**global information space**), is a major prerequisite for reducing redundancy, achieving better error detectability, greater transparency and traceability, closer cooperation and using company know-how. The targeted powerful development tools are not able to provide their features to the users without being information-technologically embedded in the landscape of other productive systems and information sources. Such extension of the informational scope of applications (see Figure 5) and engineers is critical for facilitating design-for-X, which helps to avoid loops and detours, thus accelerating changes. It also reduces the occurrence of errors from the beginning. Efficient computer-based simulation and testing are only possible if based on good information exchange with other systems. Generally speaking, upstream applications can utilize information from downstream applications to better be able to estimate the results of particular decisions (**forward view**). In contrast, downstream ProSAps can use the information stemming from upstream ProSAps to broaden their informational scope and, thus, enhance decision-making (**backward view**). And any application can reliably react to product changes handed over to it on the instance level.

Accessibility of information to anybody also presupposes **openness** of the global information space for any new system. Formerly unknown applications and information of various kinds should be integratable in order to cooperate.

The global information space should tackle both **abstract** and **specific information**: the processing of abstract information opens the way to offering and use of company know-how (see below), while specific information covers the classical PPR models and their relationships.

Information can also be divided into productive or **user data** on the one hand and **control information** on the other. Managing the control flow between applications is a means of coordinating and thus supporting cooperation including concurrent engineering. In addition, tight cooperation is only possible with a tight meshing of information. This key issue is picked up in the following subsection.

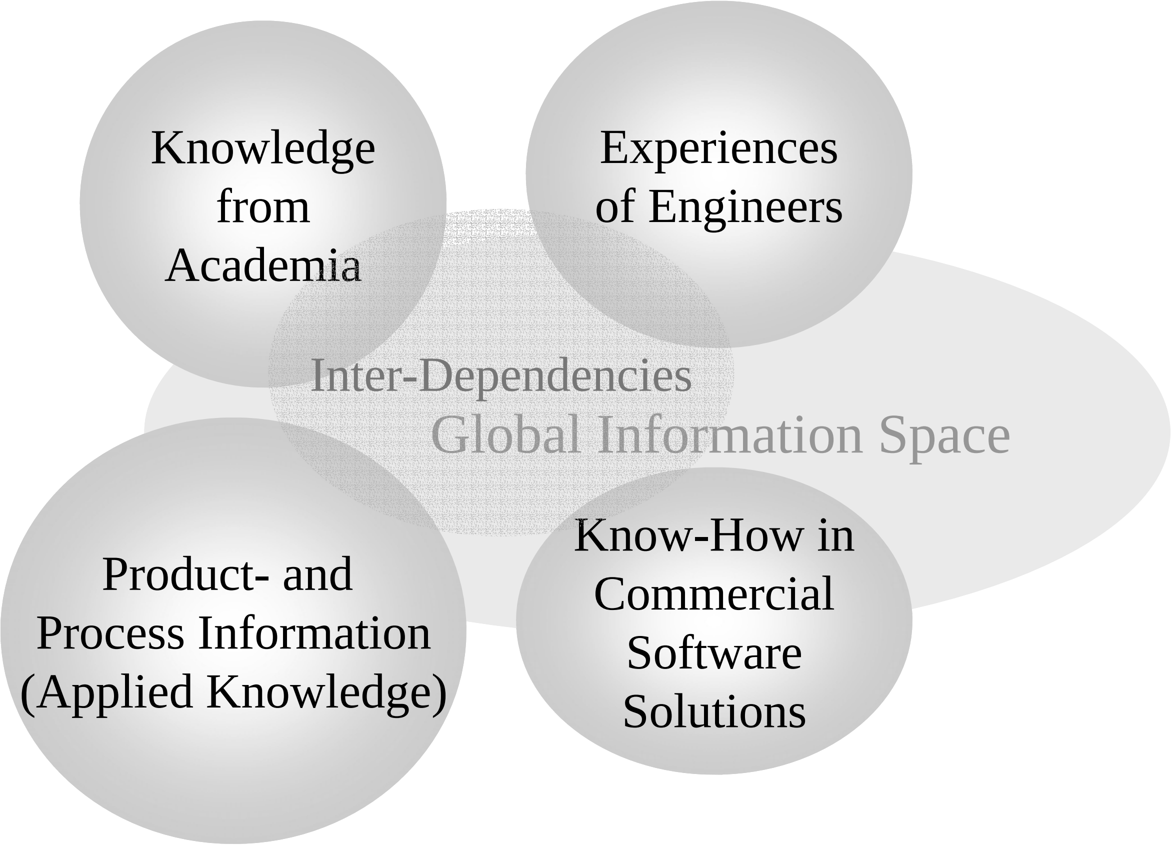

Knowledge from Academia
Experiences of Engineers
Inter-Dependencies
Global Information Space
Product- and Process Information (Applied Knowledge)
Know-How in Commercial Software Solutions

Figure 5: Knowledge Integration within the GIS

✍ The **informational scope** is informally defined here as a function of the generality and the connectivity of information (see Figure 6). Generality is equivalent to the number of cases where some information is usefully applicable: general information describes the “rules behind” whereas specific information describes the “effects only“. Connectivity is a measure for the number of cross-links between informational entities.

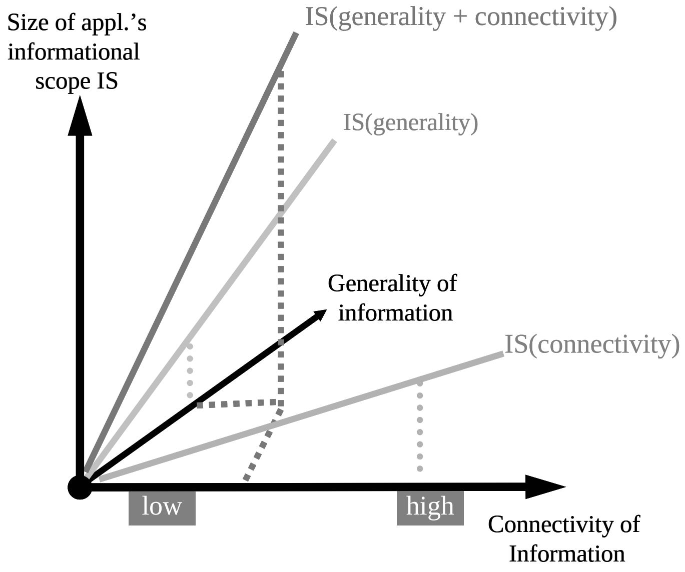

Size of appl.'s informational scope IS
IS(generality + connectivity)
IS(generality)
Generality of information
IS(connectivity)
low
high
Connectivity of Information

Figure 6: Increasing the Generality and Connectivity of Information

Relations

The importance of enabling **multi-directional associativity** between informational entities and the ability to cope with the context-dependent semantics of such active relations and the correlated concepts have been discussed in the practical analysis sections. The sophisticated handling of relations is a general key factor for high-level information flow. It is essential to recall that relations exist within the single domains but also between the different domains in product development. In particular, it is the **domain-spanning relations** that are important for the integration of domains and applications (see Figure 7).

Relations in-depth. Relations play a vital role in information processing as they represent the manifold links occurring between the representations of real-world entities within the computer. More generally, they are central elements to naturally represent links between any kind of informational entity, including relations relating other relations.

☆ For example, relations commonly group any set of informational entities for further usage in a given context, and such **constellations** frequently form the basis for appending further downstream information (by further relations). The *quality criterion* relation introduced in the section Part V – 13.3.1 below is a vivid example for such a constellation, illustrating the significance of relations.

Due to their role as significant informational entities and in order to support natural information modeling, it is useful to provide relations with own attributes.

A global information space also covers background information such as **procedural strategies** (i.e., inspection strategies, FEM meshing strategies, or machining strategies). It has been argued above that such strategies can be intuitively viewed as relations between several entities, as they represent the link between process steps. ProSAps and system services must be able to access such strategies online and interpret them at runtime. Another form of using relations at runtime is to follow **paths of relation instances** connecting product models, thus crossing application boarders.

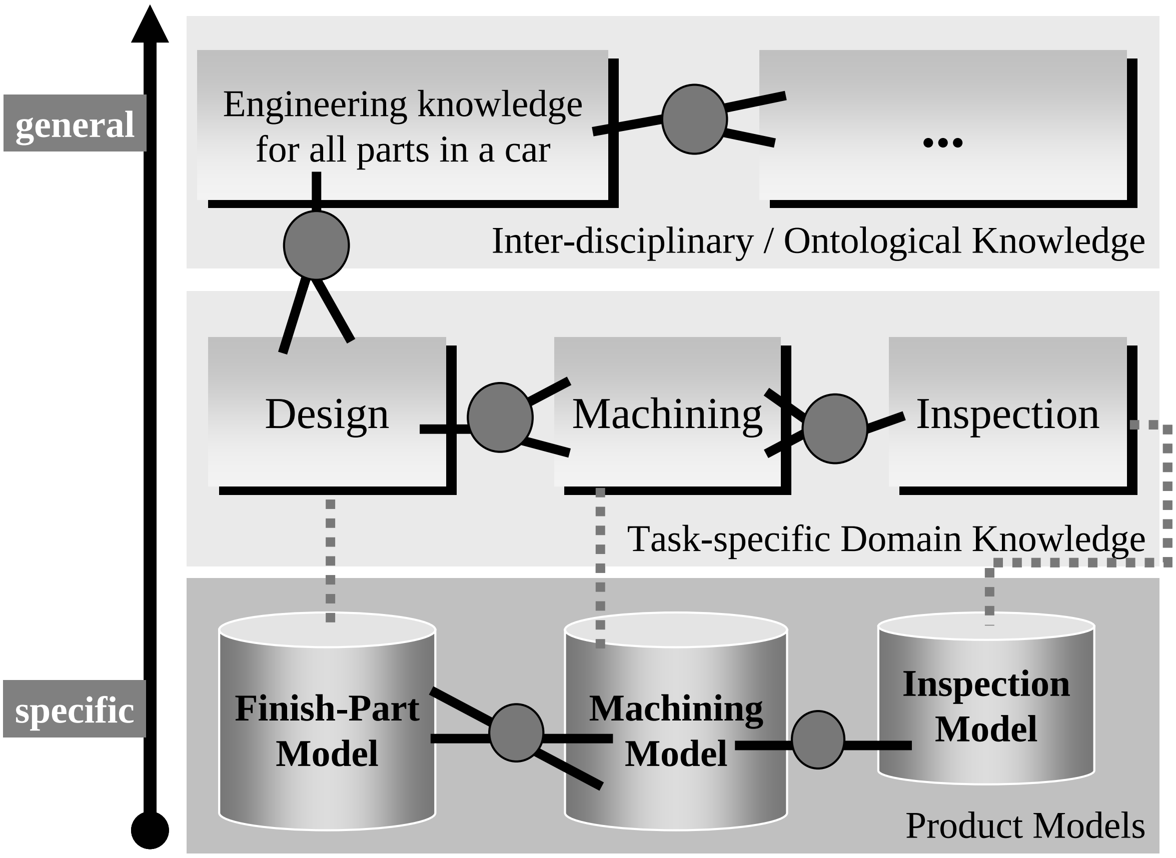

general
Engineering knowledge for all parts in a car
...
Inter-disciplinary / Ontological Knowledge
Design
Machining
Inspection
Task-specific Domain Knowledge
specific
Finish-Part Model
Machining Model
Inspection Model
Product Models

Figure 7: Significance of Relations within the Layers of GIS Information

Runtime usage of relations can and should be efficiently supported by an optimized representation for relations. This includes the support of an infinite number of relation types, and their arrangement in a dedicated meta-taxonomy in order to be able to utilize inheritance of relations. In other words, a **classification** of relations should be possible on an unlimited number of meta-levels. This allows the ProSAps to also use relations of a type derived from another one known to them. Additionally, it is an effective means for filtering information of interest at a given time. In order to be able to naturally represent procedural knowledge, relations should be provided with **methods**.

To summarize, it is suggested here that the concept of a relation be revaluated from that of a pure inter-connection of concepts to that of independent, equally important informational elements. As will be shown later on, this implies that concepts do not know about their evolvement into relations.

Contexts and Semantics

Tracking information on the **context** in which each informational entity is valid and in which it is equipped with a specific semantics is a powerful and necessary means of managing the informational contributions of the participants within a global information space, i.e., applications and engineers, and helps to avoid misunderstandings among them. Furthermore, contexts allow information to be clustered according to domains in product development, thus forming what may be termed **views** of the overall PPR information. For these reasons, a sophisticated representation of context information is considered crucial within this research.

☞ The **views** discussed here are views on the domains. They have to be differentiated from views on the informational entities (see the section *Views on Informational Entities* below).

☞ Tackling **context-specific semantics of informational entities** means to define each IE within one or more dedicated contexts and to equip it with semantics. The latter may happen explicitly by means of meta-information or implicitly by the IE's basic information type (such as *concept, instantiated concept*, or *relation*), set of attributes, and the embedding into a net of relations to other entities.

Semantics. Implicitly or explicitly represented meta-information on the semantics of informational entities may be meant for the user and/or for software applications processing it. However, any representation of semantics has to be agreed upon by humans, i.e., software engineers and users. This holds true for semantics information meant for software, as well. And, for both purposes, more or less powerful ways can be chosen. It is generally true that the amount of some IE's inherent semantics increases with the expressive power of a representation format and, in the same way, a rising amount of variance in user information is representable and usable. This means that information that relates to concepts not known at the time of creation of some piece of software handling it can also be shared.

For these reasons, it is argued that the relevant basic informational types must be reflected by dedicated representational elements within a representation formalism equipped with a formal semantics (see below). Furthermore, a certain amount of explicit representation of semantics is considered useful, which is especially true for the interpretation of relations at runtime, being of a type not hard-coded in the respective application's code.

Detail: Formal Semantics of Representation Formalisms

As concluded above, software applications are to be capable of interpreting incoming data, thus putting them into a context that is as similar to the data source's context as possible. A formal semantic specification is a prerequisite for this.

While interpreting incoming data, a computer program tries to capture their meaning from two sources. The first is the semantics of the formalism's elements. This is available through commitments of the formalism's inventors and can be hard-coded into the software. The second part of the semantics of the data is not contained inside the data themselves. It might be delivered in a package together with the user data or stay constant and be hard-coded into the program, too. The latter case is more inflexible, however, regarding which information can be transported and understood. What can a computer program achieve, if it is able to "understand" the meaning of incoming data?

~ If a computer program can recognize formal elements of the language[*] (by applying hard-coded syntactic and grammatical rules) and if it "knows" the meaning of these representational elements (REs), and consequently, for instance, the difference between classes and instances, it can process them adequately. It will be able to store them at the right place, for example, or use them to derive further associated information by exploiting the semantics of inheritance relations between classes. It will also be able to check the completeness of an instance's set of attributes. Hence, it will be able to perform a number of basic tasks linked to the "understanding" of the various sorts of REs such as objects, relations, classes, and instances. On the other hand, this means that a computer is not able to perform more sophisticated sorts of processing (see below).

~ If a computer program is to "understand" the full meaning, say, of object classes, e.g., a design feature class *Stepped_hole_A*, a language with a formal semantic description (semantic specification) is not sufficient in the general case. It would only be sufficient in case of languages especially designed to transport a fixed set of object classes. This approach, however, is not flexible enough to carry any type of engineering object emanating from any application in product development. Consequently, the second part of the semantics, as described above, is needed here.

Footnotes

[*] Used for representation formalism

To summarize, sophisticated processing of data requires formal semantics and more. This “more” is an explicitly represented description of (by definition, part of) the meaning of some other piece of information’s meaning, as defined by the information source, i.e., producer. This implies the recursive nature of the relationship between information and its meaning, i.e., the description of the meaning is also a piece of information in itself.
Thus, the representation to be applied in this work must have a formal semantic specification and should also be applicable for explicitly representing the semantics of some piece of information.

☞ The description of the meaning of some piece of information does not necessarily have to be located and represented separate from the user information. If separated, however, such a separate model will also be termed an ***ontology*** here.

Company Know-How

As demonstrated by the examples of engineering strategies, efficient **documentation** and effective **usage of company know-how** is desirable within informationally integrated applications. It should even become a vital part of it, as it helps to intensify cooperation between process participants significantly: the desired powerful development tools will make special skills and knowledge available to all employees: best practices, strategies for performing certain tasks and solving certain problems, known errors. Company know-how thus also supports the use of building blocks and leads to valuable benefits:

~ Fewer errors, loops, and detours in processes, e.g., by knowing descriptions of problems encountered previously
~ Improved error detection in products, processes, and resources
~ Less manual work by supporting automation within processes
~ Increased transparency of products, processes, and resources by providing help information
~ More reuse, less varying processes by following best practices
~ Saving and documentation of a company’s know-how so that it stays available, even if the know-how owners leave

Transparency of the global information space. Company know-how of any degree of maturity should be retrievable at any time by anyone in order to avoid redundant work and to reduce the variability of processes and products. As a consequence, also unfinished and (partly) **conflicting information** should be maintained, supported by context information. This suggests that the storage and provision of abstract information (which subsumes company know-how) should be separated from its processing in order to avoid the rejection of information entries that are inconsistent to others. See also the section *Further Theses* below. Software capable of handling company know-how and other background information must be **knowledge-based**. This implies that its behavior can be influenced by changing the knowledge base(es). Hence, future integrated ProSAps will be knowledge-based applications.

Informational Contents

The **categories of information** handled inside a global information space may be manifold. The following list cites some examples, at the same time constituting a minimum list (see also *[Dankwort et al, 1997]*):

~ Product geometry, spatial relationships, etc. (design features)
~ Manufacturing information
~ Information on the assembly of parts (assembly features)
~ Production costs for parts (cost features)
~ Quality requirements, tolerances (inspection features)

- Welding information
- Information about users (user models)
- Description of feature instances, e.g., in a taxonomy of classes including abstraction, inheritance, instantiation, handling of aggregated (compound) features.
- Mapping rules, i.e., information on transforming feature instances
- Meta-information
  - Semantics plus the attached contexts of the informational entities, e.g., semantics of various feature classes in their respective contexts such as detail design or machining planning
  - Completeness, precision, fuzziness
  - Reliability, uncertainty
  - Information source
  - Time of creation of information
  - Access restrictions
  - Messages, comments
- Consistency constraints such as rules for restricting valid attribute values
- Feature-/object-spanning background information and expert knowledge (such as general design rules or experiences, e.g., lists of known cases)
- Associativity relationships between informational entities (similarities, patterns)
- Application-specific information such as design intent
- Temporal information (also temporal sequences)
- Simulation information
- Multi-media objects of any kind

It has already been argued during the practical situation analysis that also the **non-proprietary**[*] **information sharing** as well as a **clear and structured organization** of the shared information are prerequisites for introducing significantly more powerful IT concepts, which includes a high-level information flow. The dependency on a certain openness of the vendor's software has been pointed out.

Further Theses on Information Handling

The author takes up the following theses and respective initial conclusions. Underlying all of them is the assumption that an informational integration of software applications in product development becomes easier and more straightforward, if information modeling mirrors the real situation as closely as possible and avoids changes of and restrictions/rules for real processes and software applications as far as possible. All in all, new approaches that start off from the real situation are easier to scale, better accepted, and thus easier to introduce. The goal is to handle the chaos instead of avoiding it. Basically, this means in detail:

(1) **Overlapping, redundancy and contradiction** of information chunks are typical for real-life processes and systems[†] and should not be prohibited but handled by a new solution. In fact, they are hardly avoidable effectively, especially if existing software has to be integrated into a new solution. Moreover, any clustering of information is considered to be artificial and more or less specific for a given set of applications. Each informational entity is part of the same GIS. As a consequence, there are no really independent regions of information in the general sense. Thus,

Footnotes

[*] Neutral in the sense of software vendor independence

[†] For example, due to heterogeneous software systems

clustering of information should be virtual and multi-layered and manifold at the same time. A given informational entity should be assignable to more than one information cluster, i.e., context.

(2) Any kind of information, including product, process, and resource information, generally requires the full **context**. Even product-describing geometric information cannot be regarded as a stand-alone region: it always depends on other information representing the context for this information – such as manufacturing information – and correlated by relationships to it. In turn, the same holds true, as geometric product-describing information is a vital context for manufacturing information. From this, it is concluded that, in a natural and adequate representation, information of any kind should not be strictly clustered in a sense that there cannot exist direct relations between informational entities of any cluster.

(3) A natural representation of information in the product development information space reflects the flow of development processes by providing respective sub-areas for the individual **development tasks** (and applications) and correlates them through relations representing the respective development procedures (functions, tasks). From this, it can be concluded that the representation of product development information should be highly interwoven (bias on relation modeling and management) and virtually clustered for clarity and not artificially separated by intermediate models that prohibit the existence of direct relations between informational entities of any region in the information space.

(4) Current information retrieval interfaces commonly offer set-based information access, e.g., using SQL queries. While this is also conceivable for the purposes of this research, the first and primary solution should provide **navigational access** (also called **navigation-based access**) to all the information inside the information space, as this is the typical access to product development information within current CAx systems. And it is also the more general one, as it allows the user to directly access any individual informational entity. When extending this inner focus to fields such as materials management, set-based information becomes increasingly important and, to the same degree, navigational access may become increasingly cumbersome. For navigational access, **highly expressive relations** (see also the section *Relations* above) are crucial, as they determine the steering capabilities and thus the robustness of finding relevant information during the navigation. Navigational access also calls for a **fine-grained *identification and addressing schema (IAS)*** in order to locate and identify each informational entity within a set of others, even if navigation crosses physical system boarders. Furthermore, a new IAS should allow the assignment of any ID to any informational entity without provoking collisions, as otherwise intensive synchronization between applications would be necessary.

Further details on information retrieval. In the general case, the navigational method promises to support information access within the GIS better than set-oriented calls can, as not all conditions to be put into SQL “where” clauses are necessarily known beforehand. Navigation means instead to follow relations of certain types in order to get the desired information: it is considered vital for flexible and information-controlled online communication. Nevertheless, set-oriented communication is considered sensible and worth being implemented, as it may ease one-shot retrieval and storage, where the ProSAp exactly knows all the relevant parameters of the information source or destination.

3.2.2 Automation and Reuse

Automation of routine- and infrequently performed tasks is one of the key features of powerful development tools, as it can reduce the amount of manual work, the number of errors[*], and PPR variability and boost the motivation of staff members. During the practical analysis sections, the automation of strategies has been shown to be promising.

Offering and using **pre-defined standard building blocks** for products, processes, and (partly) resources can be facilitated by the above-mentioned powerful development tools and leads to a number of advantages:

- Less variety in the products, processes, and resources
- A higher degree of optimization
- Cost-effective saving of process time through replacing several design steps by a single one and through more routine of the engineers, which in turn allows partly for automation.
- Improved information flow between process steps as information relevant for downstream processes and the related relations are already provided within the building blocks

Such building blocks can be classical **features** (see also the discussion on finish-part features above) or other objects that stand alone or form entire **constellations** of objects and relations. Large and complex constellations are also termed ***templates***.

✍ It has been pointed out that such building blocks should be specialized (and optimized) according to the tasks to be fulfilled within the respective process steps: each process step application has its own view of the PPR due to the fact that it has to fulfill special tasks using specialized technical knowledge and modus operandi for problem-solving. IT solutions must adapt to this multitude of characteristics in order to be able to represent effective and intuitive tools and aids for the users. The need for an appropriate design of features and for feature-based systems subsequently arises. Features should resemble the particular concepts relevant within the respective process steps. In other words, these feature types should correspond to the concepts the users have in mind while performing their tasks, thus assigning features the role of building blocks for creating view-specific PPR models within the perspective of the special development task. This results in specialized sets of feature types for every ProSAp. In the following, they will be termed ***domain-specific feature types***. The users of such feature-based systems take fewer mental detours when doing their job than their associates, who are forced to build their task-specific concepts from lower-level elements or have to use multi-purpose concepts that meet their needs only partially. Tönshoff and Dürr, for instance, propagate the notion of such universal features (see *[Tönshoff et al., 1997], [Dürr et al., 1997]*).

Another kind of automation already exists today. It is very common to apply **macro** techniques to automate rather simple tasks when using CAx systems. Such macros, also called scripts, are, for example, coded in the Visual Basic™ Programming Language or derivates of it. However, significant effort is required to document and maintain those macros. Efficient reuse of code is hard to achieve. It is, therefore, of interest that a new approach to informational integration of applications includes existing macros. As they contain procedural information, it is natural to deal with macros the same way as with the above-mentioned engineering strategies – in particular, because macros often represent engineering knowledge.

Footnotes

[*] Occurring due to slips of the pen during routine tasks and due to forgotten skills regarding infrequent tasks

3.2.3 Usability

Powerful **development tools** are characterized not only by their functionality but also by their usability and their degree of matching the engineers' tasks. Usability (ergonomics) arises from intuitivity of handling, comprehensibility, and tool assistance for the user. Tools optimized in this way increase the efficiency of work, decrease the probability of making errors, and help engineers to enjoy their work. Although significant progress has been made, current CAx systems still suffer from shortcomings in various areas:

~ User interface (presentation and handling)

~ Assistance (background information, advice, assessment, etc.)

~ Ability to adapt to the users' goals, skills, and preferences (today, the same user interface is deployed for all users)

~ Building blocks (objects and constellations) and methods offered to the user: users should be offered exactly the information and methods necessary to fulfill their particular task – fewer universal and low-level but more specialized, high-level, and optimized objects and routines. This suggests the usage of dedicated informational entities inside the global information space (see above) such as finish-part features for the designer – and offering views on them. Also the above-mentioned means of reuse and automation fall into this category.

~ Design-by-least-commitment should be supported: this requires flexible objects that do not have to be fully specified when first used or that can change their type (re-instantiation of classes)

3.2.4 Prerequisites for the Practical Introduction and Applicability of New IT Concepts

Automotive companies have typically invested large amounts of time and money to buy licenses for CAx software, to customize* it, and to train users. The respective software is partly declared to be "strategic" for the company, which means that no alternative software vendor's product may be used for the give purposes. In addition, daily product development is very time-critical, leaving not much time for administrative tasks. However, in large organizations, the introduction of new information technologies has a direct impact on the product creation processes as it may interrupt or interfere with the course of running projects. Consequently, it does not seem realistic to suggest to a company that it exchange its complete product development software for a new and ideal product re-inventing the world. Even small software changes lead to significant costs for training and support of engineers and for debugging. Instead, in order to have realistic chances of being adopted, new solutions should integrate legacy applications as such, thus allowing the large amount of expert knowledge behind such systems to be used. Furthermore, if adapted to the existing IT landscape, a new solution is easier to gear to the OEM's needs. For the same reasons, first steps towards integration, automation, and standardization undertaken by today's commercial software vendors have to be considered very carefully, too. As there is an increasing need and demand for pertinent solutions, automotive companies already apply these technical possibilities, although knowing quite well that they are quite a distance away from what they would actually need. But they have already made some progress such as the introduction of user-defined features in certain areas or the use of advanced tolerancing methods, and companies certainly would not be willing to do without such methods. As set out below, this results in several consequences for the design of new, ideal solutions. For example, this makes today's neutral exchange format-based solutions second choice, as they lag behind the capabilities of the original software systems.

Footnotes

* Customization covers user interfaces, data structures, and macros or scripts.

And finally, organizational factors inside companies, as discussed in section *3.1.5*, especially the very common profit center structure, may reduce the chances for funding the development and introduction of new software, even if this might lead to an overall (company-wide) optimum. Today's large automotive manufacturers have very specialized software departments responsible for comparatively small areas inside the overall field of product development. As a consequence, the services offered by new software cannot all be used by a single such department.

Consequences. Suggesting promising new IT concepts is just not enough in the automotive industry. This situation grants a much better chance of being introduced by OEMs to such new software concepts that are **scalable** to the current practical needs and able **to cope with the current software** world and – what is more – to integrate this imperfect world as far as possible. Generally, scalable software reduces costs for the roll-out, allows its benefits to be optimized for the application purpose, and broadens the possible application fields. Under the given preconditions, scalability means the ability to maintain utilization of already existing and rolled-out applications and adding new functionality on top of them by means of customization, e.g., by using programming or scripting APIs*. As newly added features directly affect the costs for customization and user training, their range should also be scalable to the actual needs.

The aspect of compatibility between new and old PPR models also plays a key role (i.e., keeping existing PPR models usable by new software). The more a new system differs from its predecessors, the more effort is required for training and PPR model conversion. For this reason, the right answer to the question of whether a new IT solution can be used in combination with the existing ones can result in large savings or additional costs. Leaving old models unchanged when using new functionality is the most elegant way to save costs and raise the software's chances of being adopted by an automotive company.

Scalability of software, in turn, depends on its **flexibility**. In general, flexibility of software can be achieved by several means:

~ Supporting configurability of software

~ Separation of program algorithms and the application's background information, i.e., adopting the knowledge-based approach. Using (a configurable amount of) company know-how is also a means of making software flexible. But this also yields two well-known major challenges: acquisition and maintenance of knowledge. Additionally, as engineering knowledge is rich in different facets, the representation format has to be very flexible as well.

Going into more detail, flexibility can be achieved, for example, by the following means:

~ A scalable degree of automation provided by the software: if new functionality comes in small steps, training costs are reduced through learning by doing. In addition, the degree of automation can be tailored to a company's needs.

~ Newly introduced knowledge-based systems can be utilized sooner, if they can cope with various complexity levels of their knowledge base contents. Comparatively simple background information can be acquired faster. Systems able to utilize it will become productive sooner. An example is information on available user-defined features.

~ Newly introduced knowledge-based systems can be utilized sooner, if they do not pre-suppose a highly filled knowledge base and are able to cope with incomplete information. Top-down filled information bases are a counter-example.

In addition, **supplier integration** is a very relevant aspect in many areas of product development, as it is quite common to out-source engineering tasks. As a consequence, technical means are required to exchange and share information with suppliers safely but, in the ideal case, on the same high level as

Footnotes

* Application programming interfaces

demanded above. “Safely” includes the protection of OEM’s company know-how to the highest possible degree. This can be achieved by managing access restrictions to informational entities (groups, policies, etc.), encryption, file-based and online communication, etc.

With respect to the representation formalism applied, there must be a good **balance between expressive power and runtime efficiency**. Unnecessary expressive power may lead to reduced processing speed, which may be more critical for the applicability of a new approach than the first issue is.

✍ In product development, efficiency of information exchange is partially critical, mainly due to the complexity and amount of information represented in CAD models, tolerancing models, etc. As it is desirable, for example, to route real inspection results back into product engineering applications, mass data are also of interest.

Strict encapsulation of informational entities. Generally, a compact representation format, primarily characterized by an encapsulation of informational entities (one representational element for one informational entity), supports efficiency of information access, maintenance, and exchange. Ideal from this perspective is **strict encapsulation**.

A central and highly critical issue with knowledge-based approaches is the problem of **how to fill information bases**. This question has to be answered by new approaches. In other words, a feasible concept for information acquisition is needed.

3.2.5 Summary – Catalog of Requirements for IT Solutions for Engineering

This section qualitatively catalogs the requirements placed on IT solutions that have been developed in section 3.2 based on the real situation in product development while considering future challenges. Details are found in the preceding sections. The catalog will be used to state the goals of this research and to formulate a catalog of characteristics for assessing existing IT solutions.

✍ Calling to mind that IT concepts and tools should fulfill as many of these qualitative requirements as well as possible in order to be able to improve product development in the above-defined sense with a maximum effect, there is, however, no hard limit for separating “good” solutions from “bad” ones. Furthermore, some of these requirements depend on others: for example, before being able to utilize company know-how and to achieve automation there has to be informational integration between applications. Although full implementation of the requirements in IT solutions is indeed very desirable, it is not necessarily practically achievable by a single research project.

The catalog lists a wide variety of applicable requirements:

(1) Means for the **informational integration** of product development applications (maintenance of a **global information space**)

~ Processing of declarative and procedural information
~ Processing of specific and abstract information including company know-how
~ Processing of control information
~ Provision of a inter-process communication between applications
~ Openness to any application and information source
~ Sophisticated representation and management of relations between informational entities (enabling multi-directional associativity, also domain-spanning; relations to relations, types of relations, taxonomy of relation types, attributes, and methods in relations)
~ Transparency of the global information space (ability to cope with conflicting knowledge within contexts)
~ Management of context-specific properties and semantics for informational entities
~ Formal semantics of the representation formalism
~ Separation of storage and processing of abstract information
~ Support of navigational information access
~ Fine-grained, GIS-wide identification and addressing schema
~ Neutral[*], flexible[†], and clearly structured representation of information

(2) Means for enabling **automation** and reuse

~ Automation of routine tasks
~ Automation of infrequently performed tasks
~ Offering specialized building blocks such as features or object constellations
~ Management of macros

(3) **Usability** of the software by the engineers

~ Task-oriented optimized concepts and routines (e.g., domain-specific feature types)
~ Provision of assistance for the user
~ Sophisticated user interface (presentation and handling)
~ System adaptivity to the users
~ Support of Design-by-least-commitment

(4) Prerequisites for the **practical** introduction and **applicability** of the new approach

~ Ability to cope with and utilize legacy software
~ Compatibility of new and existing PPR models (instance information)
~ Configurability of software
~ Knowledge-based software
~ Ability to cope with varying complexity of information in the knowledge base
~ Ability to cope with varying amount of information in the knowledge base
~ Support of supplier integration
~ Representation formalism: balance between expressive power and runtime efficiency; encapsulation of representational elements for informational entities
~ Feasible concept for information acquisition

Footnotes

[*] Non-proprietary
[†] Through high expressiveness plus semantics

Chapter 4 Problem Description

This chapter states and motivates the goals of this research work and condenses them into the description of the research question.

4.1 Goals of This Research

This section states the goals of this work (detailed problems) as a motivated selection from the preceding chapter's catalog of requirements for IT tools.

As a consequence of the restricted resources available for this research work and considering the dependencies between individual catalog elements, this research focuses on specific goals, while carefully bearing in mind that the unlisted ones are not endangered. During the discussions above, informational integration in a GIS turned out to be the most central requirement, on which most of the others such as automation depend and which yields significant benefits.

For these reasons, this research aims at developing an IT concept for facilitating the **informational integration of applications in a global information space** (primary focus, catalog item no. (1), including all sub-items) and the **automation of routine tasks** (secondary focus, catalog item no. (2), including all sub-items) in automotive product development. In order to insure that engineers and software applications get the information needed, their informational scope is to be widened and isles of information are to be fused by embedding the applications into a global information space. Design-for-X can be achieved on this basis. As pointed out above, the information may consist of declarative and procedural components as well as specific and general information.

Automation and specialized, pre-defined **building blocks** help to make processes and product components more standardized, avoiding errors and detours and reducing costs incurred due to product changes. The concept worked out here is to provide a foundation for this as well as for intensive **use of company know-how** and for **improved usability of systems** (catalog item no. (3), especially the two first sub-items, viz. optimized concepts and routines and provision of assistance for the user). Since this work's results should be of use for product development processes in the automotive industry, also the above-given **practical prerequisites** (catalog item no. (4), all sub-items) are to be met by the new solution: for example, scalability and flexibility are to insure the future applicability of the productively used and very costly legacy software applications.

A **software prototype** will demonstrate the practical applicability.

Further Motivation of the Goals Selected. In addition to the motivation set out in the preceding sections, the selected goals are also considered to be the most urgent ones in terms of a need for process improvement in practice. This assumption is supported by the efforts made in the software industry – however, the results have been argued to be insufficient above. On the other hand, finding answers to these challenges promises to yield significant benefits for automotive OEMs.

Furthermore, the solutions to the stated problems seem to become more realistic nowadays, and the chances for bringing scientific concepts into real application can be judged optimistically. The main reason for this is that OEMs apply more powerful CAx tools than available a few years ago. Current software is object-oriented and customizable in terms of concepts and algorithms.

4.2 Research Question

The **general problems** motivating this research work can be formulated as the following **research question**:
How can a global information space be realized that is practically usable in product development and that supports automation of routine tasks?

☞ Due to the heterogenity, complexity and dynamics of the product development domain, a general and theoretical validation of concepts is not possible. Therefore, several practical scenarios from product development will be used as references for the validation of the suggested solutions, i.e., to decide whether they realize a GIS that shows the above-described properties to a degree that is sufficient and beneficial within the individual scenarios. Benefits are always to be compared with the costs.

The **detailed problems** are represented by the goals of this research stated in the preceding section.

Part III – State of the Art

The goal of this part of the thesis is to investigate the state of the art in those areas of information technology directly or indirectly relevant to this research's targets. Such solutions will be considered, giving the best answers to any combination of the above-stated challenges by showing a good balance of costs and benefits. While the common practically applied solutions have already been discussed in the section Spotlighting Product Development at a Major Automotive Manufacturer, *the section at hand captures approaches not (yet) applied for tackling the above-stated goals. The first chapter in this part of the thesis sets the foundation and guideline for these investigations by providing a catalog of characteristics of IT solutions.*

Chapter 5 Catalog of Characteristics of IT Solutions for Engineering

The following systematic catalog of characteristics has been abstracted from the above catalog of requirements for IT solutions for engineering and will be used as a guideline for identification and discussion of relevant approaches during the next chapters.

This catalog is to help in making the discussion of approaches more uniform and, thus, the approaches better comparable. Nevertheless, it will not be fully applied in each case in the sense that each catalog element will appear in each approach's discussion. Instead, only such aspects will be picked up for the individual approaches that promise to contribute to the goals of this work. Judging whole fields of scientific research or of technologies for relevance is superordinate to the discussion of individual approaches and will also be based on this catalog.

✍ The author refrains from motivating the single catalog entries, as the benefits for the reader would be marginal. Principally, such issues have been admitted as suitable to judge each approach as to whether they meet the individual requirements of the catalog set out in the section *Summary – Catalog of Requirements for IT Solutions for Engineering*. Existing catalogs have been considered, e.g., OntoWeb's[*], but have been found to be too rough-grained, in general.

✍ Refer also to the appendix *Coherent Glossary of Important Terms* for the terminology applied.

Catalog

(1) **Focus** of the approach: tackled domains (potential application fields)

(1a) Approaches geared for engineering; e.g., multi-purpose IEs vs. task-oriented optimized IEs; is concurrent engineering supported?

(1b) (Others such as company information management)

(2) **Information handling**: modeling and processing of information
What kind of information is or can be covered? (expressiveness of formalism used); reasoning support

(2a) Orthogonal basic information types:

(2a1) Specific information (facts, instantiated concepts + relationships) vs. general information (conceptual = terminological information = classes of concepts + relations, rules, constraints, constellations = building blocks; company know-how); abstraction methods

(2a2) Declarative (static) information vs. procedural information: propositions, sequential procedures = strategies, including control structures (loops, cases)?

(2a3) Informational entities: domain objects vs. relations between domain objects vs. relations between relations: yes/no and which entities?

(2a4) Meta-information (e.g., context information, explicit semantics, modality)

(2a5) User information vs. control information (e.g., events)

(2b) Basic information structures (how is information represented within computer models?)

(2b1) Clustering of models and information bases, including the underlying motivation? partitions = physical clusters, meta-information = virtual clustering; name spaces; task-specific optimized concepts?

Footnotes

[*] Ontology Environment SIG of the OntoWeb initiative, see the URL *http://delicias.dia.fi.upm.es/ontoweb/sig-tools*

(2b2) Basic approach: logic-based (declarative information, facts, universal and existential quantification, implications=rules, distributed-&-extendable concept descriptions) vs. frames (declarative information, weak/semi-encapsulation by special role relations) vs. strictly encapsulated OO approach (object-defining properties vs. external properties; all object-defining properties within the same representational element)
(2b3) Representational elements used for informational entities (concepts, relationships, rules, procedures, building blocks, meta-information, events). Examples:

- ~ Concepts: do they have properties (attributes, relations), methods?
- ~ Relationships: do they have properties or methods? Same as for concepts? Classified (typed) relations? Standard (multiple?) inheritance relations for concepts? (Multiple?) inheritance of relations (meta-layer)? Arity, role names, cardinality, interfaces; Multi/uni-directional relations? Inter-domain and/or intra-domain?
- ~ Strategies: How is automation (mapping[*]), knowledge (strategies) modeled?

(2c) Information processing/reasoning: is it possible using the given formalisms; are there services offered or suggested?
(2c1) Reasoning
(2c2) Automation

- ~ How and by whom is automation knowledge processed?
- ~ Automation on demand vs. automation on the fly
- ~ Generation and maintenance of links between (mapped) concept instances?
- ~ Direct or indirect automation relations?
- ~ Direct or indirect mapping?

(3) Handling of **application-spanning information** and facilitation of communication
(3a) How are informational entities uniquely identified and addressed?
(3b) How is communication achieved/coordinated? Is there a global **information space**? What are the characteristics? Examples:

- ~ Architecture: which applications are involved? How do they co-operate?
- ~ Online (= inter-process) or offline (= exchange files) communication?
- ~ Information retrieval: set-based or navigational or proof-based?
- ~ Openness
- ~ Transparency
- ~ Neutral / proprietary?
- ~ (Expressiveness, semantics)[†]
- ~ How are user data represented and exchanged? Are neutral formats such as XML or STEP used?

(3c) Which methods are applied for understanding others’ information? Examples:

- ~ Common data dictionaries
- ~ Semantics
- ~ Contexts

(3d) Is control information processed? (Facilitation of **cooperation)**; Who processes it?
(3e) Which methods are suggested to achieve supplier integration?

Footnotes

[*] Automation can be considered equivalent to mapping. See below.
[†] Important here, but already covered above

(4) Is there user **assistance** or other means to enhance usability?
(5) How are **scalability** and **flexibility** supported? Examples:

- ~ Configurability
- ~ Knowledge-based software?
- ~ Is existing software re-used? Is backward compatibility to existing PPR models facilitated?
- ~ Is the system usable with varying model contents?

(6) Filling the information bases: are methods of information acquisition suggested?

Chapter 6 Overview of the State of the Art

This chapter identifies research fields and technologies that are potentially able to contribute toward achieving the goals of this work and is arranged according to them. The above catalog of characteristics of IT solutions serves as a guideline for this and for the surveying and discussing of individual approaches in Chapter 7; the result is a set of relevant groups of approaches.

☞ Due to a lack of space, the following survey and discussion cannot explicitly cover all the references that have been considered within this research. Thus, approaches that are either less relevant or not at all material to the topic or those lying beyond the scope of the research will be excluded from discussion within the current chapter.

6.1 Goal: Informational Integration

Informational integration of software applications is generally desired in almost any domain, regardless of the kind of information that is to be processed. Consequently, the approaches are numerous and manifold. This invites a look at non-engineering solutions also, which will be done in the following. Generally, the goal can be reached more easily in cases where applications of the same origin are to be integrated or where applications meant to work together are being developed in parallel. As the major software vendors in the engineering domain have a tradition of several years of software implementation, they are hardly in the position of developing several cooperating software systems from scratch. Hence, typically, also vendors of all-in-one solutions have to cope with pre-existing conditions and/or backward-compatibility.

When scanning existing application-integrating solutions, the question arises whether software for the engineering domain differs fundamentally from other domains' software in a way that prevents non-engineering solutions from being adopted. Taking the overall product development into account and considering the logistic process chains covering a variety of information this assumption is rejected here.

✰ For example, information is processed describing material and non-material properties, mathematical and geometrical models, requirements placed on the product, processes and resources, or covering economic issues and that materializes in small or large numbers of informational entities.

Thus, it is assumed on the contrary that solutions from computer science are generally also well applicable for the engineering domain.

6.1.1 Relevant Groups of Approaches

Based on these general insights, groups of approaches that are potentially suitable for contributing to the issue of an informational integration of applications will be identified in this section.

Approaches to model self-contained information (group 1, *SCI modeling*) are certainly important sources of stimulation since an adequate method to formalize, represent, and manage self-contained information can be considered crucial to achieve the goals of this work. This is reflected in the individual criteria in the catalog of requirements for IT solutions.

A variety of ***standardization approaches*** (group 2) targets an application-neutral data exchange, thus focusing on the representation formats (group 2a, *exchange formats*) to be used on the one hand and the means to facilitate communication between running applications on the other (group 2b, *inter-process communication*). As the flow of information is a key aspect of informational integration, methods of the groups mentioned will have to be explored.

To acquire input on the issue of which classes of systems are to be involved within an integrated systems network and which are to be their individual functions and properties, it seems sensible to consider existing approaches to ***integration architectures*** (group 3), i.e., software concepts and systems, aiming at the facilitation of some form of information flow between two or more software applications or between them and the users. The latter are also called *Information Systems (IS)*, the former could also be termed *integrative architectures*.

Evaluating approaches to manage potentially large amounts of complex information (knowledge) within databases such as the rule-based **deductive databases** lies beyond the inner scope of this thesis; it is, however, subject to future work.

In the following, the groups of approaches introduced above will be examined in more detail.

6.1.2 SCI Modeling

This section identifies such approaches to SCI modeling that can be expected to contribute to answering the question of how to represent information within a global information space adequately. As motivated above, such information should include abstract and specific information as well as procedural and declarative information. Furthermore, meta-information such as semantics and contexts are equally as important as the actual user information since it is typically managed by productive applications such as CAx systems.

Several sub-fields of **artificial intelligence** (AI) research are traditionally concerned with the representation and processing of what is commonly termed *knowledge* in AI and what is termed *complex information* in this thesis: expert systems (knowledge-based systems), knowledge representation, neural networks. Although **logic programming** is not subordinated to AI, logical representation formalisms are suited to represent declarative complex information. All of these fields provide formalisms that are suitable to model self-contained information. However, **earlier knowledge representation languages** such as KRL, KL-ONE, KRYPTON, AROM that were extensively discussed in the 1970s and 80s will not be discussed in further detail here, as the lessons learned from them are reflected in current solutions mainly relying on XML syntax and additionally frequently provided by improved relation concepts and addressing schemes.

Classical **expert systems** have proved to be generally hard to maintain due to their unstructured, often rule-based, knowledge bases. Today's knowledge-based systems often take so-called model-based approaches, using frame- or object-oriented information models. Their structure arises from relations between concepts. Model-based approaches are employed not only for representing the meaning of concepts (ontologies) but also for representing some kind of technical system in order to simulate its behavior, e.g., for detecting reasons for malfunctions.

In other fields of computer science**, ontologies** are also used to describe the meaning of concepts employed within some piece of information. ***Semantic Web*** technologies belong to these application fields. Procedural complex information is, of course, represented by most programming languages. Although this is not self-contained information, programming languages can provide input on the more general design issues of representation formalisms.

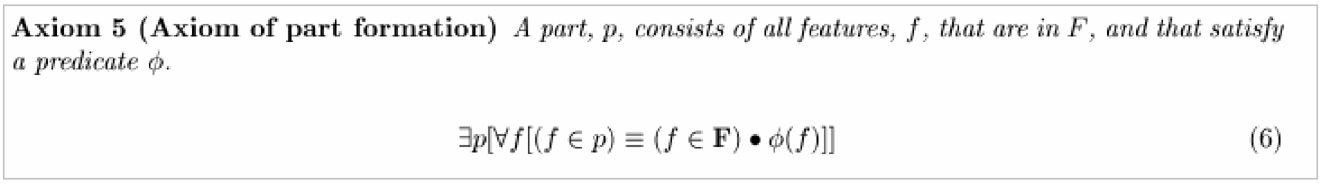
The Semantic Web is also an application field of **knowledge management** (and of the first generation, also termed *knowledge capture*), which deals with capturing complex information inside computers. Not in all cases is the managed information self-contained. *Engineering Books of Knowledge* (**EBoKs)**, are document management systems commonly offering a user interface on a hypertext basis. Some of them, also called *Advanced Books of Knowledge (ABoKs*), strive for a semantic integration of the engineering documents' contents. To do so, retrieval-relevant concepts and their relations are modeled within an ontology and thus semantically more clearly. Additionally, correlated or identical document contents become obvious and usable. In the following, individual EBoK or ABoK systems are not discussed, however, for the sake of general ontology-based approaches.

Also not discussed in further detail will be approaches targeting the construction and upkeep of **thesauri** such as the *WordNet* project by the Cognitive Science Laboratory of the University of

Princeton, which targets a rich online thesaurus for the English language (see *[Miller et al., 1993]*). This will be motivated briefly here: these approaches target a specific use case of their internal models, namely the typical use case for thesauri, e.g., finding synonyms and antonyms. Although WordNet applies representational elements that are generally applicable in SCI modeling, e.g., concepts (*nomen*), concept-defining[*] relations (*descriptive adjectives*), or concept-spanning[†] relations (*relational adjectives*), it loses its generality on providing thesaurus-specific representational elements. The latter are, for instance, dedicated to the representation of verbs. To sum up, thesaurus approaches do not offer generally applicable insights, as provided by ontology-based or other approaches to SCI modeling.

Also not considered more closely will be the **semantic networks** (see the glossary) although conceivable for being used to represent semantics, they do not show formal semantic specification. Furthermore, they are – at least in their pure form – suitable for capturing simple information contents only.

Not considered is also the approach taken by Filippo Salustri due to its logic foundation[‡] (see *[Salustri, 1996]*). Salustri defined a “formal theory for knowledge-based product model representation”. He characterizes his approach as follows “*…AIM-D, the Axiomatic Information Theory for Design, a theory able to describe the structure of designed products in a logically rigorous manner. It is not a product modeling system in itself, but rather a logic of product structure whose axioms define criteria to determine the logical validity of any product model.*” To make this approach more tangible, Figure 8 illustrates an example formula including existential and universal quantification.



Axiom 5 (Axiom of part formation) *A part, p, consists of all features, f, that are in F, and that satisfy a predicate ϕ.*

$$\exists p[\forall f[(f \in p) \equiv (f \in \mathbf{F}) \bullet \phi(f)]] \tag{6}$$

Figure 8: Logic-based Representation of Features in a Part [Salustri, 1996]

Object Centering as a Basic Communality in Knowledge Representation

This section gives a rough impression of basic affinities between so-called knowledge representation approaches using the object-oriented paradigm’s object centering as an integrating element.

One of the most straightforward approaches to describe parts of the real world is to describe the collection of objects that can be identified and that are most relevant for a given problem. **Object orientation** (in short, ***OO***; see also the appendix *Coherent Glossary of Important Terms*) promotes this idea by structuring information representation and processing along individual entities of the domain instead of using data structures distributed over program algorithms that are organized along system functionality. It is exactly these issues that yield the key benefits of OO: (1) better structurability of software code (data storage and algorithms are grouped around specific relevant

Footnotes

[*] That is, object-defining

[†] That is, external

[‡] As motivated above for expert systems and below for rule-based approaches in general

concepts = objects of the domain) and (2) a reflection of the real world's domain, which seems to be more natural to most programmers. Consequently, the resulting models of the domain are – as any model is – specifically suited for a given set of problems and interests.

While objects in the OO sense may include declarative and procedural information or knowledge, most symbolic **knowledge representation formalisms** may be deemed compatible with the idea of arranging declarative information around objects or entities of interest, i.e., a sort of ***object centering***.

Frame-based representation formalisms. Frame-based representation formalisms have a rather long tradition and are widely spread to cover a variety of problems. Frames are the basis for rather universal but not very efficiently processable representation formalisms and do not offer capabilities for the representation of procedural information[*]. However, frames do provide object centering (the objects relevant for frame-based approaches are concepts). A frame-like representation does not principally differentiate concepts from their attribute fillers[†] – the values of attributes are again other concepts. This semi-encapsulation approach produces straightforward information structures but slows down data storage, processing, and retrieval.

✍ It also does not seem easy to be maintained down to all individual instances within any product model. All natural and floating numbers, for example, would have to be unique individual frame instances and would have to be referenced by all other instances whose attributes are set to the respective values. This would mean managing a single instance "4" of the frame representing natural numbers.

Frames, as OO does, stress the relevance of the concepts represented by the frames, while relations between them play a sub-ordinate role. The frames' sub-optimal efficiency behavior originates in the fact that IEs are not strictly encapsulated. It has been argued (see Part II – 3.2.4 above) that this is sub-optimal for achieving this research's goals. Existing frame-based formalisms are, anyhow, well-worth being discussed, as they may provide hints or solutions regarding very relevant issues of SCI modeling such as identification and addressing of informational entities, handling of meta-information, handling of contexts, and integration of separated information models.

Predicate calculus (first-order logic) actually talks about object classes (concepts) when using *universal quantifiers* and real world objects when using *constants*. Relations of any arity can be expressed in the predicate calculus, too. E.g., inheritance can be easily modeled by means of *implications*. If/then-like, so-called **Production Rules** are a subset of the predicate calculus.

Conceptual graphs, **entity relationship diagrams** etc. describe classes and instances of objects and their inter-relations. This includes modern ontology-based systems using frame- and logic-based approaches to explain their semantics. Even sub-symbolic methods such as **neural networks**, read input and produce output that can be interpreted, for example, as object properties. However: objects in this sense are not equally strict encapsulated, as OO objects are.

Typically, knowledge-based systems are able to represent also relations other than inheritance in order to express a domain's knowledge and thus going beyond the OO methodology. Such relations are also of importance for integrating different domains in product development. As strategies often link domains, also procedural knowledge should be representable within relations. This idea, in turn, extends many knowledge-based approaches, which often abstain from using procedural knowledge corresponding to OO methods.

Summary. Several characteristics of object-orientation have already been argued in this thesis to be of interest for this work: object centering, means for procedural representation, strict encapsulation. This section made clear that because object centering is commonly accepted in knowledge

Footnotes

[*] Although there have been approaches to extend them by procedural attachments

[†] Called role fillers

representation, the pertinent approaches are in principle compatible, and even integratable (see also *[Zimmermann, 1994]* for an example of a hybrid representation system). However, due to their varying targets, the approaches most often do without procedural elements, and also the degree of encapsulation strongly varies. Also, object-orientation in its plain form is not enough: relations should be more powerful than OO's built-in ones, which are simply pointers.

From this, it can be concluded that there is no real one-way decision for and against a suitable representation formalism, as there will almost in every case remain the option to add further representation capabilities by taking over ideas from different formalisms. This does not mean, however, that any approach is suitable for any task, as the specific characteristics may in fact exclude certain solutions from being adopted in this research.

Logics and Sub-symbolic Representation Formalisms

Some ontology definition languages allow usage of **logical formulas** (*statements*) as part of the concepts' description, and logical representation languages in general are one discrete means of SCI modeling. Logical formulas are very powerful in that they may represent complex propositions inside a single statement: inside the same statement, information on more than one concept or object, and general and existential quantification, and logical operators combining several predicates may be found, including inference rules (*implications*). The other way round, information on a specific entity may be spread over multiple statements. This makes retrieval and transport of information on specific entities more difficult and makes such knowledge-bases hard to maintain. This issue appears to be even more relevant, as the product development domains quite commonly are concerned with individual entities (engineering objects and their relationships); PPR models consists of encapsulated instances with assigned attributes and this information has to be retrieved and shared and updated. It is hard to see, how individual feature instances could be reliably updated, if the information on them is spread over several, possibly complex logical statements. It is also hard to manage to be sure to retrieve all information defining a given feature class, if the class's definition might be spread over several statements, possibly situated in several information pools – it does not seem feasible to require all specific information in a global information space to be stored within a single storage location. Furthermore, logical formulas are collected in un-structured sets where they can be complemented by (negated) formulas to be proved and (eventually) processed by reasoning algorithms.

This missing structuring has been proved to be a real bottleneck in practical application. Although the expressive power of logical languages is a grave but commonly the only grave argument to apply them in today's solutions, it is in turn exactly the reason for the author's decision to exclude logical languages from being considered inside this work as candidates for representing the informational key structures within a global information space: the author considers the expressive power to be not necessary in product development and the complexity of statements hinders maintainability of information inside a global information space. Such complex logical statements can be compared to unnormalized databases in that they do not separate information according to the (instantiated) concepts or relations it refers to. Nevertheless, there are some approaches to SCI modeling, using logic-based ontology definition languages that are discussed below in order to get insight into their properties.

✍ These considerations do not imply that logical formulas may not be applied within some other information structure (yet to be developed in this work) for dedicated purposes requiring exactly this high expressive power.

✍ The assumption that logical expressive power is not needed has to be finally proved in practical applications, which is only possible to a certain degree within this work.

☞ The decision to primarily not consider logic-based representation formalisms does not mean that the author has abandoned the possibility to reason on represented information.

Representation languages are symbolic as they use literal symbols to denote some concepts or individuals or relations within the domain. Sub-symbolic formalisms such as neural nets, encode information following principles that do not allow mapping of the domain's entities to the model contents by one-to-one associations. Even, if an assignment could be made at a given time point t_1, it will likely to get lost at another time point t_2. This means that sub-symbolic formalisms do not use a certain set of symbols to denote a certain set of domain entities.

Ontologies – the state of the art in SCI modeling?

In a number of scientific approaches and an increasing number of commercial software solutions, **ontologies** (see also the appendix *Coherent Glossary of Important Terms*) are viewed as the answer to the challenge of SCI modeling. However, there is not "the" ontology and unclearness in wording is common. As a consequence of the widely differing properties of the used representation formalisms, there are many ways to define what is termed an ontology.

So, the following sections will discuss representation formalisms with formal semantic specification, but also broader approaches, supplementing a framework for arranging and clustering informational contents of different kinds.

As this work targets a global information space including not only information on the meaning of some other information, but also this other information itself – which may include general and specific information such as product models of a CAD system, and procedural information such as inspection strategies – ontological approaches will presumably not be able to (efficiently) cover all of the issues stated in the catalog of requirements in a degree desired here; this can be assumed regardless of details in expressive capabilities, which will have to be analyzed below. Furthermore, they typically do not strictly encapsulate object properties (see section Part II – 3.2.4). So, although they cannot directly meet all the requirements in this work, they can provide insights on many SCI aspects. For these reasons, both ontological and non-ontological approaches for representation formalism will be included in the following discussion.

6.2 Goal: Automation and re-use

Architectural issues are crucial for matters of automation in product engineering as well as for the re-use of components of engineers' work results. For this reason, the above-mentioned group 3 of ***integration architectures*** is relevant here, too.

Furthermore, there are many approaches to ***integration by automation*** (group 4), whose respective methods are potentially applicable to this work's purposes. This is especially true for the sub groups of ***feature linking approaches*** (group 4a), while so-called ***knowledge-based design systems*** (group 4b) will well be considered, but not be discussed in more detail in this thesis.

✍ Knowledge-based design systems typically use (a) rule-based formalisms for representing their "knowledge", or (b) procedural scripts, or (c) a combination of both, where rules are partially integrated into the scripts. All of these options allow for powerful representation and also powerful automation. However, they show the draw-backs already stated above for rules, which is the ultimate reason for not discussing these approaches in more detail here: the most prominent of them is the severe lack of maintainability[*]. Two scientific approaches falling in this category are the knowledge-based design of spur gears by Siegmar Haasis, and the expert system of Jaluria and Lombardi describing the design of thermal equipment and processes (see *[Haasis, 1995]* and

Footnotes

[*] This is equally true for both rules and scripts.

[Jaluria & Lombardi, 1991]), while CADFEM, YVE^{TM} and CATIA^{TM} Knowledgeware^{TM} represent commercially available knowledge-based design software.

6.3 Goal: Optimization of Usability

Representative for ***approaches to enhance systems' usability*** (group 5), two sub groups shall be mentioned: solutions from the field of *user modeling* (group 5a) and *attached user interfaces* (group 5b). The **user modeling** research community cares about techniques based on and utilizing models inside a software system that are to reflect and simulate properties, capabilities, wishes, etc. of humans sitting face-to-face to a computer. The phrase ***attached user interfaces*** shall denote complex software systems, whose programmability is utilized to simplify their usage by pre-defining and guiding typical and difficult task sequences, rarely performed by the users.

✍ As optimization of usability and especially the research field of user modeling seems to be highly promising for the author’s future work and as its influences on this research seem to be well enclosable, the employment of the corresponding techniques will be not further discussed within this thesis (see also future work). However, this research’s goal of providing a basis for optimization of usability will still be maintained and kept in mind during the development of own solutions. Related arguments will be given.

6.4 Goal: Meeting Practical Prerequisites

Scalability and flexibility are basic properties of software systems that cannot be seen isolated from others. The group of approaches to ***integration architectures*** (group 3) already brought into the range of vision are also of relevance with respect to these matters.

Feasible method for information acquisition. The lack of ideas and solutions for information acquisition, harmonizing with and seamlessly fitting into the product development process, is one of today’s major problems in information technology – and will probably be one of tomorrow’s most urgent ones.

Knowledge discovery approaches (group 6a) try to extract information out of large packs of data and ***natural language processing*** solutions (group 6b) do the same based on text blocks and both do not involve human interaction. Also the group of ***user modeling*** (groups 5a and 6c) solutions, already introduced above, promises interesting results when trying to get information from a computer user such as an engineer, because such systems can rely on their user model to help them interpret and supplement direct user input.

✍ As for the optimization of usability, the issue of identifying **feasible methods for information acquisition** is also of special interest and highly relevant, but also well-enclosable. Therefore, and due to limited resources, the deployment of the corresponding enhanced techniques will be not further discussed within this thesis (see also future work). However, basic techniques will be extensively discussed and solutions suggested.

Chapter 7 Survey and Discussion of Identified Groups of Approaches

During this chapter, representatives of the groups identified in the overview of the state of the art will be shortly introduced and discussed.

7.1 Group 1 – Representative Approaches to SCI modeling

For receiving a more representative overview, this section discusses a heterogeneous set of approaches to SCI modeling.

7.1.1 Topic Maps

Focus. Although topic maps may not be mentioned first when talking about frame-based approaches, they carry some of the most significant properties of frames. Topic maps are located amongst the ambitions to catch the meaning of data retrievable over the Internet (see also Semantic Web below), but are also applicable more universally to describe the meaning of some piece of information by describing the meaning of its components (informational entities). They have been ISO-standardized not too long ago. In the context of this work, topic maps can provide a good basis for discussing fundamental properties of SCI representation formalism. For an introduction to topic maps and *XML topic maps (XTMs)*, please refer to section *Part VII – 23.2*.

Basic information types. It is possible to represent specific and general information such as individual instances or classes of concepts and individual relations or types of relations. Context information is represented through *scopes*. XTM *types* can be compared to non-encapsulated frames: they do not possess procedural methods such as OO classes, and their properties are defined via external relations (practically incompatible to existing OO systems, resulting in a lack of efficiency; representation of procedural engineering strategies is not possible).

Representational elements. The concept of a *topic* inside a topic map is a rather general one; in fact, almost anything in the real world's domain as well as an IT system's model of the domain might be declared a topic. This implies that all these entities are represented by the same representational element named *topic*.

As *types* can be typified as well, it is possible to create types of types, that is, any number of abstractions for each topic. This matches frame or class taxonomies connecting elements by inheritance relations and is thus a very common and useful feature (reduction of redundancy leads to more clearness and better maintainability of a model).

The **semantics** of individual topics is not represented explicitly but is to be derived implicitly from its characteristics and scope (see below). Although this is a common solution, it sets aside a chance to tackle the semantics problem partially by a straightforward solution such as the specification of semantic-defining attributes combined with pre-defined value ranges.

Each topic comes attached with a single *scope* as a kind of meta-information specifying the **context** in which it is valid. The need to support context meta-information has already been pointed out in the section *Informational Integration of Applications* above. However, the path taken by XTMs has to be investigated to more detail: XTMs allow the definition of several topics referring to the same subject (e.g., domain object) if these topics are assigned to different scopes. This resembles multiple views on a given object. However, although these *subject references* are useful in the Internet domain to locate resources, in the product development domain they at the same time refer to class **instances** from within classes – if the topic represents a class. A CAD system would have to know about all the feature instances it had produced so far. This does not seem a desirable solution in this case. If the topic represents an instance, this is not the case; yet this would still mean that the same object would be described by multiple sets of attributes, potentially without the existence of correlated classes. In fact, also in the general case of IT systems, the XTM subject references appear as a specialized solution and the problem of locating a certain instance of a class should rather be taken out of the classes' descriptions itself and solved by an appropriate management of instances and their relationships.

Also, it does not seem sensible in the general case of integrating software applications and the models generated by them to allow instances to stem from multiple, but unrelated classes. Rather than knowing about their common instances, two concepts abstracting the same "thing" from different perspectives should be related to each other by appropriate relations on the class level.

In order to achieve a flexible **context** management, the following solution seems more suitable: the same concept can be attached to multiple contexts at a time. As a context can be considered to define a view on the domain, this allows common concepts to be shared between several of such views. For example, detail design and machining could share some features that differ from each other, for example, but carry the same names. If the same domain objects are abstracted by multiple concepts, they will consequently be represented in the IT system by multiple instances each that should be inter-related.

XTM allows the use of multiple **identifiers** for a topic. One of them is the *base name*, which has to be unique within a given scope. This solution seems to be feasible but could be enhanced by adding meta-information to the individual names allowing decisions to be made as to which name to use in the single situations. **Properties** (*characteristics*) of topics and relations are represented by means of an *occurrence*, a *topic name*, a *subject identifier*, and a list of *roles* (for relations only). In addition, some representational elements in XTM may carry a fixed list of XML attributes that, however, do not correspond to classical attributes of concepts. Attributes in the classical sense are expressed through the relations that a topic is involved in (topics are not encapsulated). Relations do not have attributes at all. *Occurrences* refer to information sources described by a given topic or relation and thus are not relevant to any class in the general case. In fact, they are specific for XTMs' original focus. A general solution could utilize a regular attribute instead. Furthermore, in the general case, it seems more beneficial to refer from instances to classes rather than from classes to instances, as this solution saves effort in maintenance.

Topic names and subject identifiers have already been discussed above. *Roles* are a natural means for documenting relation partners' function within a given relationship.

Each XTM **relation** (*association*) carries a type, which is specified by a topic. Thus, relations may carry non-encapsulated characteristics represented by further relations. The appropriateness of managing characteristics of relations and of typifying relations has been argued above. The option to abstract relation types in multiple layers, allows clear directories of relation types to be created.

However, relations cannot carry methods. This fact leads to the necessity of defining abstract data types on top of the language in order to represent them naturally. Such additional representational elements are not supported by any built-in functionality of XTM parsers, however. The same holds true for the small number of built-in elements for covering meta-information: for example, context information is restricted to the specification of rather rudimentary but hard-to-manage scopes as sets of topics. Also, the implementation of a more sophisticated **addressing** schema reflecting the typical architecture of cooperating software applications in a common information space calls for solutions on top of XTM. Furthermore, the consequences of missing **encapsulation** have already been discussed above.

The shortcomings from the perspective of this work that have been discussed so far basically originate in the differing goals. XTMs' possibilities for describing information sources on the World-Wide Web can be regarded as sufficient. However, there is one drawback that is of a more fundamental nature: XTM employs the same **representational elements** (topics) for expressing both concepts and relations. This allows representation of propositions of certain kinds rather elegantly (meta-information on relations is part of the same knowledge base and thus can serve as meta-information and, at the same time, as regular content of the knowledge base). On the other hand, due to the smaller amount of inherent semantics, it makes information harder to understand and maintain than it would be possible with dedicated (more specialized) representational elements. For these reasons, it seems more promising to structure the fundamental types of informational entities in a more small-grained fashion, thus also achieving a higher percentage of machine-processability. Types of concepts and types of relations should be represented separately; meta-information on relation types should be represented within the relation types themselves. The object-oriented interface solution could be applied to describe role fillers more flexibly instead of specifying specific concepts. This solution retains the (desirable) equality in treating concepts and the relations between them and the possibility to also express relations between other relations.

To summarize, XML topic maps introduce a series of features that can be enhanced further and combined with others to meet some of the requirements of this work on representation formalisms.

7.1.2 Manufacturing Integration Based on Information Management

Focus. The doctoral thesis published by Eric Lutters suggests a solution for the control of manufacturing processes based on their current information needs and the currently available information (see *[Lutters, 2001]*). It also introduces a solution for the informational integration of applications, as this is regarded a prerequisite for improving process control.

Yet, in earlier papers, Lutters adopts the idea of equipping information with explicit semantics by applying so-called ontologies (see *[Lutters et al., 1998]*, *[Lutters & Kals, 1999]*, *[Lutters et al, 1999]*): *“Instead of merely exchanging data, it is preferred to have access to meaningful representations of the existing information, reflecting the current state of affairs [Kals &Lutters, 1998]. This is emphasised in recognising that the main input and output of most design and engineering processes is information.”* Lutters and Kals presuppose for their new approach, concisely called ***Information Management (IM)***, *“that information generated by the separate departments (in a company) is attached to an overall and widely accessible model”*. In this case, departments can “pull” desired information. From this, they conclude IM's central thesis, namely that *“the focus can be on the information in support of the control of the manufacturing processes, and for this reason, the course of the manufacturing processes may be guided by the use of, and the need for information.”*

Basic information types. Ontologies carry abstract information explaining the semantics of concepts occurring within the available concrete information. While this information is declarative by nature, the derived process control information is procedural. Basically, concepts, relations, and their respective instances are handled. Relations correlate concepts and their instances but no other relations. Rudimentary meta-information is also managed for the informational entities mentioned.

Basic information structures. Eric Lutters advocates an axiom termed *Independence of Information*, which references and extends Suh’s functional independence axiom, which is part of the *axiomatic design* method (see *[Suh, 1990]* and *[Suh, 1999]*). Lutters’s new axiom promotes the logical separation of information sets according to the domains affected. Overlapping, redundancy, and contradiction are to be avoided. Consequently, information on products should be stored separately from process information, for instance, and is considered to be independent from it. Thus, the information base is pre-clustered into three pools called *information structures*, which are affiliated to orders, products, or resources, respectively. Orthogonally to this, Information Management defines the basic information types *control information*, *manufacturing instructions*, and *context information* (see below for some details). *Domains* cluster down each information structure: for instance, two domains within the product information structure could be *geometrical product definition* and *functional definition*. To complete what has been introduced above, Lutters suggests maintaining a set of ontologies within a *metabase* (see Figure 9) in order to document the meaning of facts and to define their physical storage location as well as access rights. The *semantic ontology* consists of an *ontology of state* and an *ontology of transition*. It hosts concepts and relationships between them. A concept’s meaning is defined by its individual situation within the ontology and within the information cluster it is assigned to.

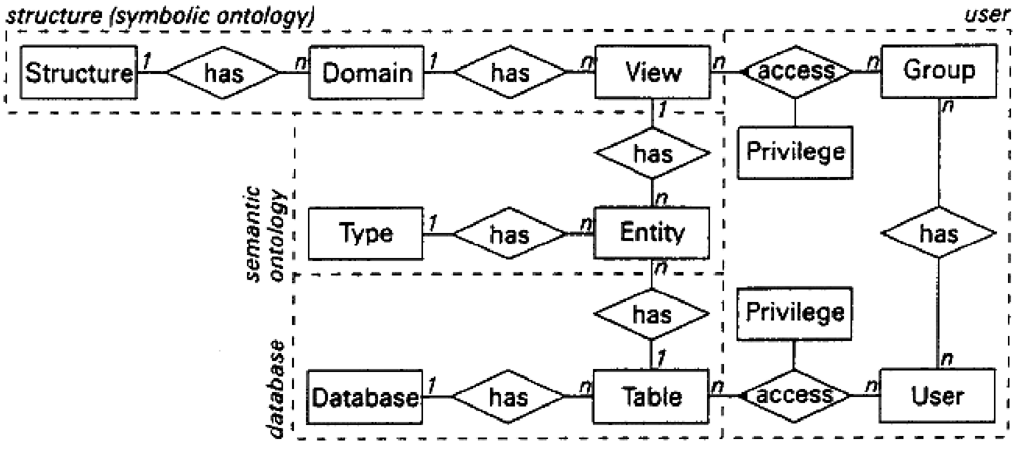


Figure 9: Arrangement of Entities in the Metabase [Lutters, 2001]

The *symbolic ontology* logically groups the concepts in the semantic ontology and the corresponding facts into separated *views*, *domains*, and *structures*. This triple defines an informational entity's context. The *ontology of transition* uses dedicated relation types to describe the sequence in which instances of concepts can serve as informational input for other instances of concepts posing as information drains. This is commonly not found in other ontology-based approaches and provides the foundation for a generator system to derive process control information. The method shows similarities to that of finite state machines in computer science: it enables derivation of a detailed sequence of all sub-tasks (from rough- to fine-grained) that are necessary to satisfy a certain information need. Both the information need and the sub-tasks are represented by (or linked to) specific concepts. Each reflects the sub-task's need to retrieve an instance of the respective concept and the actions to be taken to fulfill it. An example is the relation *production plan needs as its input process plan*, which can also be denoted as *production plan(process plan).*

Eric Lutters employs *elements* and the *relationships* between them as his **basic representational elements**. He thus follows the Tichem and Storm, who considered them suitable for an appropriate representation of product structures (see *[Tichem & Storm, 1996]*). To specify the properties of elements and relations in more detail, Lutters takes up the *conceptual graphs* approach propagated by Sowa (see *[Sowa, 1992]*). Such graphs consist of *concepts* and *conceptual relations* (see *[Sowa, 1984]*). Concepts and conceptual relations can occur on the specific and on the abstract level and are represented in Information Management using identical data structures. They can be equipped with (concept-defining) properties by means of attribute relations. These signalize a semi-encapsulated and frame-like representation that does not principally distinguish objects from their attribute fillers (the fillers of attributes are again other objects; see also Part II – 3.2.4 for encapsulation). Both concepts and relations carry a fixed set of meta-information crystallized in the respective database table structures. Lutters proposes that apart from the *type* and *identity* fields, a database table should host the context information triple *view* + *domain* + *structure* and a multi-purpose attribute *"aiding users or applications in the exchange of signposting information on e.g. the completeness, accessibility or maturity of an element".*

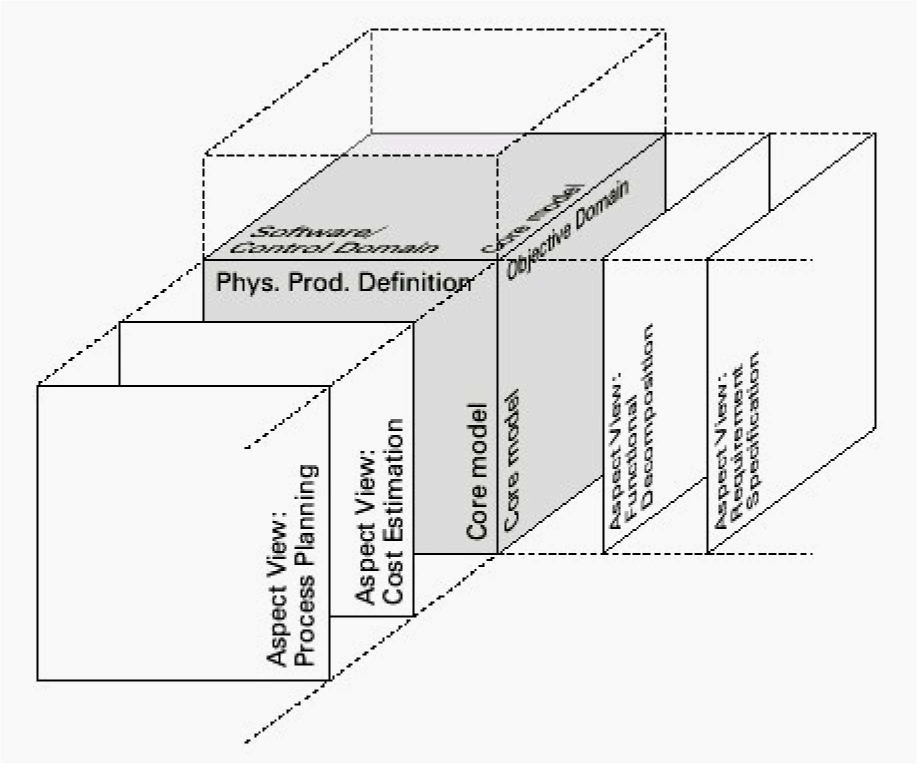


Figure 10: Basic Representation of Lutters's Product Information Structure (PRIS) [Lutters, 2001]

Resulting from a specific set of goals and boundary conditions, the further theses made in this research (see the section *Further Theses*) deviate notably from what has been set out regarding Information Management. As the former promotes management of redundancy and inconsistency of information rather than avoiding them, Lutters's new axiom and the conclusions derived cannot be adopted. Instead, it calls for a single information space where each informational entity can be assigned multiple contexts. Resulting from this, again, there is no dedicated clustering according to order, product, and resource. Furthermore, it is not possible, based on the prerequisites of this work, to enforce the usage of unique concepts for each given class of domain objects, which is the result of Eric Lutters's dedicated information structures and which is indeed useful for the assignment of ontology-of-state contents. It becomes inadequate, as well, to define a certain information cluster such as geometric product description to be the master and others such as the functional view to be views on it. Figure 10 displays Lutters's core model in the center of the product information structure. Instead, information representation as desired in the *Further Theses* means managing an information space whose elements are densely interwoven and where views come up by filtering given types of informational entities from throughout the information space. As a further result of this research's claim to handle the chaos of heterogeneous information processing, its representation formalism must provide even more expressivity in dedicated respects than that of Information Management. The latter's is ultimately motivated to allow the derivation of process control information according to information needs. That is especially true for contexts, meta-information, and relations: to keep track of the validity, versioning, etc. of informational entities, it is considered important in this work to provide an element within the representation formalism that will serve as a basis to carry an extendible set of meta-information. The availability of a fine-grained representation of relations is crucial within this research (again, see the section *Further Theses* above): an application-spanning management of relation instances is necessary for the integration of instance information but, at the same time, is not provided by existing applications. Therefore, such relations have to be maintained from outside the ProSAps. Sophisticated relation-based integration, in turn, requires that relations have to be equipped with additional meta-information on their semantics*. In particular, it is very useful for expressing a relation's semantics to specify the role of a partner within a relation using a role name that is different from the role filler's type. The classification of relations should be possible on an unlimited number of meta-levels, i.e., a taxonomy of relation types is desirable, as discussed above. Furthermore, relations should additionally possess the ability to refer to other relations. A strict encapsulation of all informational entities has been motivated in the foregoing, e.g., for reasons of efficiency of access.

Application-spanning information handling. For storing and retrieving information, Information Management uses set-based expressions, finally translatable into SQL statements. As pointed out in the section *Further Theses*, this work's primary solution should provide navigation-based access to all the information inside the information space. This fact further emphasizes the need for a fine-grained representation of relations and of the addressing scheme. Nevertheless, set-oriented communication has been argued to be sensible and worth being implemented also in this work's context, as it may facilitate one-shot retrieval and storage in those cases where the ProSAp exactly knows all the relevant parameters of the information source or destination, respectively.

On top of the information structures and ontologies sits an Information Management **infrastructure**: a software system offering a collection of tasks that *"can be used as a basis to initiate, accompany, control and evaluate all the manufacturing processes in a structured and transparent way"*. The current Information Management system consists of a collection of libraries to be linked into each application. In order to use the system functionality, ProSAps call Information Management functions from those libraries. There is no central Information Management application at the moment. In Information Management's current approach, each application manages its own set of databases including the *metabase* and has to handle the synchronization with other applications by calling API methods. Although information pushing is declined in IM for reasons of reducing the amount of useless communication between applications, an application can also provide information to others that are pulling it via the Information Management API. For this purpose, a dedicated IM interface class has to be implemented. Lutters suggests as future work that a server-based solution be implemented here, mainly in order to achieve a higher degree of integration in terms of synchronization. Referring to a server-based solution, he notes: *"This implies that all user applications act on the same conceptual graph, distinctly reducing the problems with changing information in applications."* and *"In future research an Information Management application will therefore take an important place"*.

Filling the information bases. As pointed out above, IM's information base embraces instance information and abstract information including task-based state-transition information. The abstract information is created the moment a concept type first appears through one of its instances. Lutters denotes this construction of general information as a posteriori and bottom-up. For the purposes of this work, this approach seems hard to handle in its ideal form, as the targeted applications typically need to exchange an arbitrary amount of information at an arbitrary degree of maturity of the PPR models. This fact presupposes an already filled concept model†. As a result, the author propagates a mixed, bottom-up and top-down approach; thus, abstract information should always be defined before the instantiation of the concept (top down). Bottom-up in the author's terminology means to build up abstract information for certain software applications while the taxonomy layers situated above it are only filled rudimentarily at that time. Further motivation behind this is the assumption that the integration of new entries into the concept model and their correlation to existing concepts should be – at least in the general case – done by a human who understands the full context: it should not be done

Footnotes

* The *quality criterion* relation, illustrated in Figure 4 is one example for this importance of relationships.

† This term is used equivalently to *information model* and Lutters's *ontology*

looking at a single application only. However, if this assumption is accepted, it is not possible to perform this action efficiently each time a new type of object comes up. Since this research assumes applications to be known a priori and to pre-exist[*] when creating abstract information, it seems promising to target the rather constant set of concepts and relations that each application potentially wants to exchange with others. This set also depends of what is considered optimal from an overall view of the product development process.

Summary. Eric Lutters developed the Information Management approach within the same universitary context as the author. Nevertheless, the goals deviate: this research strives for informational integration only, applied to applications along the product development process chain and without further defining the usage on top of it. This deviation of research objectives is followed by a considerable deviation in the proposed choice of solutions such as the representation formalisms. Information Management focuses on the derivation of process control information from given states in the information bases. The respective representational properties are different from those of a solution for the integration of pre-existing applications, especially as addressing schemes, contexts, and relations need to be more fine-grained in this research than they do in Information Management. IM establishes a common understanding of concepts employed in the different domains. Based on this, it is able to append and to process control information. Yet, this work aims at a co-existence of multiple terminologies, instead. However, it seems possible to integrate both approaches, as an approach for integrating applications seems, in a sense, more basic than an approach for process control.

✍ Information Management is a generally applicable approach and has been adopted and further elaborated in several consecutive research works at the University of Twente. Two of them – described in *[Mentink, 2004]* and *[Wijnker, 2003]* – will also be mentioned in this work. As they have been carried out more or less in parallel to this research, they will be briefly examined in Part VI – Chapter 18 *Update on the State of the Art*.

7.1.3 Pragmatic-Situative Knowledge Representation

Focus. Rainer Ostermayer suggests a semantic framework (in German, *semantisches Rahmensystem*) for achieving a *pragmatic-situative knowledge representation* (see *[Ostermayer, 2001]*).

Basic Information types. Ostermayer's knowledge management methodology supports acquisition and usage of general and specific information. Assigning explicit context information to each informational entity is considered the crucial method to achieve an information representation, which is significantly more effective than other ontology-based approaches are, in terms of finding the right (specific) information to satisfy a user's need. All information considered is declarative. Concepts are distinguished from relations between concepts – both on general and specific level. Relations do not involve other relations, although they can be grouped to form more complex relations. Control flow is not in the focus of Ostermayer's work. His key targets, both relying on the use of contextual meta-information, are (1) to allow computers to assist their human users in combining pieces of information by discovering similar information contents (semantics + pragmatics) and (2) to ease information retrieval. Target (1) is partly motivated by the wish to expose new use cases for existing engineering solutions. Amongst others, Ostermayer's emphasis on context-dependency of information matches with requirements for future IT solutions stated above in this work and motivates the following discussion.

Ostermayer recognizes several factors, contributing to a statement's pragmatic-situative meta-information and influencing the way it should be modeled within a computer: the information content of the statement, the recipient, the intention of the sender, and the time period during which the statement is valid; in other words: the questions "what has to be communicated to whom?", "for

Footnotes

[*] In the sense of "not newly developed together with a new approach for informational integration“

which purpose?", and "within which time period?". This urging to represent dedicated **meta-information** seems also correct and necessary in the product development field, which shall be illustrated by the following brief discussion: the **intention** of some information's sender could be translated into a ***domain****'s engineering tasks*, i.e., a step within the development process. More detailed information would presumably not be necessary as it is common background knowledge in engineering. The specification of some information's **recipient** would specify a dedicated recipient within a given domain, which could be (a) the storage location of a certain model or (b) a certain software application. This ***locator*** information could, together with the specification of the **application**, detail the *domain* information further. These considerations will to be worked out into more detail in the section *Identification and Addressing Schema in the GIS* below.

Basic information structures. Ostermayer identifies the **de-contextualization** as a principal drawback of existing representation formalisms including programming languages: individual statements loose much of their context, inside which they are valid. In order to solve this problem, Ostermayer suggests that the **meaning** of statements should be represented explicitly. While this approach is shared also by other ontology-based approaches, he adds context information to further specify, where this meaning applies (context-specific semantics). As again typical for ontology-based systems, a concept's meaning is represented indirectly by its embedding within the ontology. This ontology is supposed to provide the semantic background for and explanations of the contents of software applications' data models, committed to it.

Ostermayer suggests structuring the ontology by distinguishing several types of elements stored inside: he defines specific **concepts** describing *Sachverhalte* and hosting a set of other concepts, describing objects from the domain. While this set of domain-object concepts paraphrases the meaning of the respective Sachverhalt concept, the Sachverhalt concept, in turn, represents part of these domain-object concepts' **pragmatic-situative context**, in addition to the above-mentioned types of meta-information. This solution reflects the insights that domain-object concepts can only be well-represented if supplemented by context information, and that not any concept can be easily represented using concept-describing properties. While the author shares the first insight, he chose a deviating solution for complex informational constellations such as Ostermayer's Sachverhalte. The goal to integrate various software applications can be achieved easier, if information structures are kept simple; a concept with properties can be retrieved and transported more efficiently than a concept that consists of other concepts, describing it. Moreover, it is assumed that the information to be shared in product development can be adequately represented by the classical concept representation, which is also used inside almost any existing software application. Therefore, usage of only one basic kind of concept is proposed together with one basic kind of relation.

☞ This decision still allows the definition of an unlimited number of concept and relation types in the sense of respective classes. It restricts, however, the number of basic information types for concepts and relations to one each.

For grouping concepts, utilization of relations is proposed. For interrelating such groups, other relations shall be used. Thus, the elegant way to paraphrase the meaning of concepts is sacrificed for reasons of efficiency and simplicity.

Basic representational elements. Rainer Ostermayer's Sachverhalt concepts are typically arranged in a partonomy. Thus, the ontology is logically clustered by means of Sachverhalt concepts. Each domain-object concept may be correlated to others using properties (*Merkmale*) represented by relations. These properties are similar to slots of frames. Relations may represent ***concept-defining***[*] properties (*attributes* in the OO terminology, typically represented by specialization relations) or ***external*** properties (defining another portion of a concept's context, thus called *relational properties*

Footnotes

[*] That is, object-defining

or *contexts* by Ostermayer). Ostermayer deliberately suggests this non-strictly encapsulated representation in order to support the detection of common properties between concepts: if two or more concepts have an overlap in their sets of external property relations, a certain degree of similarity or connaturality can be assumed. This benefit is additionally supported by the specification that relations may span Sachverhalt boundaries. It is, however, not considered within this work.

Ostermayer selects an appropriate representation formalism following this train of thoughts:

- An **object-oriented** formalism allows the representation of specific and general information (*real* and *abstract* propositions in Ostermayer's terminology). **Inheritance** allows for a modular representation and a re-use on different levels of abstraction.
- **Constraints** relating to contexts should be represented by (logical) axioms, as these directly influence all elements within a context.
- **Logic rules** are an adequate means of representing directed constraints on object level. Additionally, rules enable hypotheses to be set up for validation by reasoners.

Ostermayer chose the *Ontolingua*[*] formalism for his prototypical implementation. Ontolingua is based on the *Knowledge-Interchange format (KIF*[†]*)*, which in turn is based on *LISP*. Ontolingua is a logic-based, frame-oriented formalism with semi-encapsulated concepts.

✍ *KIF* is a representation language, developed by the *Interlingua Group* of DARPA's *Knowledge Sharing Effort* for the exchange of knowledge between systems. This use case is supported by the fact that KIF allows referencing statements within other statements and thus adding meta-information such as modality and origin. KIF's information models can be modularized and reused inside other models. It is supposed to become an ANSI standard *[ANSI, 1998]*.

✍ Ontolingua's representation formalism is based on KIF and eliminates some of its drawbacks. For example, it adds **slots**, thus supporting a frame-like semi-encapsulation of representational entities. Ontolingua has been developed at the *Knowledge Systems Laboratory (KSL)*[‡] of Stanford University (see *[Ontolingua, 1997]* for a tutorial and related publications). The principle implementer is James Rice. Ontolingua includes a **server-based system** *[Farquhar et. al., 1997]* for the management of ontologies. The KSL website states: "*Ontolingua provides a distributed collaborative environment to browse, create, edit, modify, and use ontologies. The server supports over 150 active users, some of whom have provided us with descriptions of their projects*". Duineveld summarizes "*The Ontolingua server provides a repository of ontologies, allows ontologies to be created and existing ontologies to be modified.… After completion of an ontology, the ontology can be added to the repository for possible reuse. …This repository consists of a large number of ontologies from different fields. …An important aspect of the Ontolingua system is the ability to design an ontology collaboratively. This allows different users from all over the world to work together in constructing a single ontology. In order to be able to do this, the server uses a notion of users and groups*" (see *[Duineveld et al., 2004]*). Although KIF is a language developed to support knowledge sharing, it does not include commands for knowledge base query or manipulation. For this purposes, Ontolingua implements *OKBC* (see also section *Protégé-2000 and OKBC)* below for more details) which is complementary to KIF.

Pro's and con's of rule- and **constraint**-based approaches have already been discussed above. Ostermayer propagates their use for **generally**[§] **valid propositions**, hardly ever supposed to be changed. In respect to Ostermayer's targets, this approach is well arguable – within this work, it will

Footnotes

[*] See the URL *http://www.ksl.stanford.edu/software/ontolingua/*

[†] See the URL *http://logic.stanford.edu/kif/kif.html*

[‡] See the URL *http://www-ksl.stanford.edu/*

[§] Entire ontology

not be adopted, however, for the reasons already given. It is suggested, instead, that context-wide constraints are represented explicitly as a set of relations between IEs. In order to ensure strict encapsulation of representational elements, it is further suggested that **meta-information**, including modality, is represented directly inside the elements instead of using referencing techniques, as possible in KIF.

✍ As already suggested for Sachverhalt relations, also in the currently discussed case, the direct and explicit representation is favored. Object-internal constraints can be represented within the objects themselves using formulae, as concepts are considered to be atoms, i.e, the smallest entities in the information space, connectable through relations.

For the same reasons, Ontolingua and KIF are also not considered suitable to represent other propositions in the context of this work.

Ostermayer's second means to indirectly cover the semantics of concepts are **relations**[*], which is common for ontology-based approaches. Each of his relations has a dedicated type and can relate two or more concepts, i.e., Sachverhalt concepts or domain-object concepts. Relations do not carry attributes, other than the type, arity and the above-mentioned meta-information. As argued in the section *What is needed? Deriving Requirements for Future Product Development IT Tools* above, **typification** of relations is considered crucial also within this work, and it is suggested to revaluate the **concept of a relation** even more from that of a pure inter-connection of concepts to that of an independent, equally important informational element. This means to give them (relation-type-dependent) attributes and to correlate also relations by relations. Such revaluation becomes necessary in this work, due to the targets and presuppositions, chosen for this work.

Ostermayer proposes a classification of types of relationships. In this taxonomy, Ostermayer specializes the uppermost relation type *conceptual relation* (in German, *Begriffsbeziehung*) into *hierarchical* and *non-hierarchical* relations. These are further distinguished into *sequential* and *pragmatic* relations, those into *abstractions* and *compositions*.

✍ Ostermayer's taxonomy of relation types can provide a good basis for use in product development, even if the by far biggest number of relation types can be expected to be, what Ostermayer calls *pragmatic relationships*.

Application-spanning information handling. Ostermayer defines Sachverhalt concepts to serve as **name scopes** for the contained domain-object concepts and their properties, thus avoiding the need for unique identification of IEs all over the knowledge base (ontology). This meets the requirements of product development, too, as existing development steps commonly use at least partially unique concepts, while at the same time, terminology may overlap orthogonally. This means that different concepts may be assigned the same identifiers and vice versa. As other ontology-based approaches do, Ostermayer implicitly assumes an at least logically **centralized** system **architecture**, where it is not necessary to **address** informational entities across system borders. Nevertheless, he makes certain steps into this direction by defining relevant meta-information for IEs such as recipients and duration of validity, as discussed above. As hinted at, Ostermayer proposes a method to ease **information retrieval**. For this purpose, the ontology can be filled with a base vocabulary, stemming from the concepts needed within the targeted application domains. Furthermore, the ontology represents the way, concepts are constructed from more basic concepts according to DIN2330. The most specific concepts, contained in the ontology are those that are expected to be used within data and information models used by the domains' software applications; the most basic concepts are those, expected to be found in fuzzy information retrieval queries of users. Thus, the ontology spans the gap in-between the user queries and the vocabulary of the software applications, and allows translation of fuzzy user

Footnotes

[*] Relations are concepts, again, but of a higher arity

queries into queries that can be directly answered by computers. On the way down from the user queries to the domain-specific application queries, the system needs contextual (pragmatic-situative) information to remove the fuzziness from the original query. Some of this missing information is retrieved from the intra-propositional context (other concepts within the same statement/query), the rest has to be retrieved from outside, partially from the application's domain, partially from the user.

Filling the information bases. The other way round, Ostermayer suggests to acquire information by scanning such existing natural language documents, which already use a semi-formal terminology such as DIN and other standards. In step two, based on the partially filled information base, it seems easier to scan other documents, e.g., from the World-Wide Web, while "understanding" words by disassembling them and mapping them onto the concepts in the information base.

To sum up, Ostermayer identifies a series of issues, also of relevance for this work. These are the importance of contexts and name spaces, the use of typed relationships together with the proposition of a basic taxonomy of relation types, and the differentiation between concept-defining* and external properties of concepts. However, due to his diverging focus, Ostermayer's solutions turn out different from the ones, suited for this work. Details have been stated above.

In the meantime, Rainer Ostermayer works in the same company as the author does, and the question arises, whether his approach might be integratable with the one covered by this work. And, comparable to an integration of Eric Lutters's Information Management approach, also an integration of Rainer Ostermayer's *pragmatic-situative Knowledge Representation* seems conceivable, and basically for the same reason. However, in Ostermayer's case, this remains to be proved, as the solutions for context representation deviate, which might have consequences for the appropriateness and naturality of the representation.

7.1.4 Semantic Web Technologies

A significant number of scientific approaches claims to be or strives to be compatible to the Semantic Web efforts: this could probably be denoted as the state of the art in the field of SCI solutions. The key idea behind the Semantic Web approach is to explicitly describe semantics of information chunks from the Internet by means of ontologies. *OWL* is a language used to formulate the contents of an ontology and to correlate them to others. *XMI* allows ontological contents to be exchanged by explicating necessary meta-information.

Resource Description Framework (Schema), RDF(-S)

✍ RDF and RDF-S are introduced in the appendix.

Focus. *RDF* is a representation formalism for resources in the World-Wide Web. *RDF-S* provides a type system on top of it. Neither is designed to be employed as an ontology definition language. For this purpose, *OWL* was developed on top of them. OWL will therefore be discussed in more detail than RDF(-S). However, as OWL relies on RDF and RDF-S, and as it shows some of their characteristics, this will be taken as a topic in this section.

The philosophy behind RDF schemas considers RDF-S types to be just one piece of information on an object amongst others. This does not correlate with the perspective dominating this research, i.e., that each object is an instance of exactly one concept within the same context. The latter is motivated by the handle-the-chaos axiom and the fact that each model is just a partial reflection of reality, dominated by a unique view on it. Computers manage model contents and not physical objects, and it

Footnotes

* That is, object-defining

is not considered intuitive to view any instance as being the same as the physical thing it represents. Thus, two instances in two models can never be identical. If it should be of relevance that two instances describe the same thing, this should be made explicit by using relations.

Basic informational entities of RDF-S are *classes* of WWW resources, *instances* of them, and *relations* between them. Instead of WWW resources, any other object could also be represented. Resources as well as *properties* can be arranged in inheritance hierarchies. A resource can be an instance of more than one class. While the benefits of inheritance relations between concepts seem to be commonly accepted, typing and inheritance of relations are obviously not. However, this has been stated to be useful in the context of this work.

Basic representational elements. RDF directly represents only binary **relationships**. N-ary relations have to be broken up into several binary ones, which is unintuitive and reduces clearness of representation (see also the conclusions made in sub-section *Bottom-Up Interviews with Body-in-White Experts* of Part V – 13.3.1 below). So-called *blank nodes*[*] (see Figure 11) are only a workaround to handle n-ary relations rudimentarily.

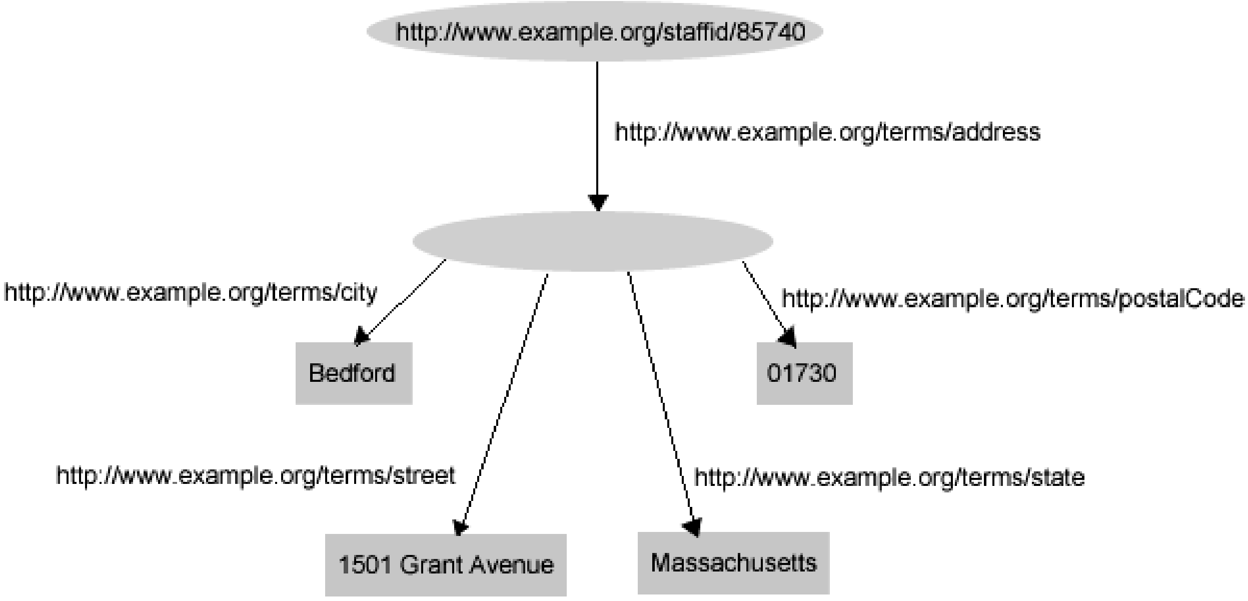

http://www.example.org/staffid/85740
http://www.example.org/terms/address
http://www.example.org/terms/city
http://www.example.org/terms/postalCode
Bedford
01730
http://www.example.org/terms/street
http://www.example.org/terms/state
1501 Grant Avenue
Massachusetts

Figure 11: Blank Node for n-Ary Relations [W3C RDFprimer, 2004]

✍ The blank node workaround must be considered sub-optimal as blank nodes are perceived to be unnatural, lack semantic clearness, and reduce processing efficiency. For example, it is not clear whether a relation partner is really a relation partner if it represents a property of the relation itself.

Application-spanning information handling / Addressing schema. RDF's notion of uniquely identifying each individual entity over the complete information space meets the needs of this research, i.e., managing an information space shared by multiple applications.

URI schemes are addressing schemes used to address dedicated types of entities, namely instances of specific concepts. There is no central synchronization of addressing schemes. For the purposes of product development, it seems promising to provide an addressing scheme that is standardized up to

Footnotes

[*] Also called anonymous resources

the point where individual applications manage their informational entities internally. This means that addressing information should identify and address domains and applications using a standardized meta-information structure, leaving space for application-internal addressing of informational entities. It also means that domain and application IDs are managed centrally.

RDF offers single-string addressing. A *URIref* consists of two parts. From what has been stated above, this can be considered sub-optimal compared to a more deeply structured addressing schema.

Web Ontology Language (OWL)

✍ OWL is outlined in the appendix.

Focus. OWL has been developed to allow the meaning of web contents to be represented using ontologies. Its **basic informational entities** are concepts (called *classes*), instances of concepts (*individualizations*), and the relations between them (*properties*), which resemble specific and abstract information. OWL employs data types as fillers of *datatype properties*. Such data types may be RDF literals or XML schema data types (see *[W3C XMLschema, 2001]*). Analogous to other logic-based formalisms, OWL is a declarative language. It provides meta-information on relations between classes such as cardinality and symmetry. In order to allow automated processing of information between applications, it seems highly beneficial to employ a standardized set of simple data types. When using XML as the basic representation formalism, the XML schema data types are a natural solution.

Information structures. Concepts are not encapsulated, although there are frame-like data-valued properties. Relations do not correlate other relations. Relations do not carry attributes (properties). As to the meta-information OWL provides for its relations, cardinality and role names are also useful in the product development domain, although the latter are implemented only sub-optimally due to the lack of n-ary relations. Further, the specification of object-oriented interfaces seems to be much more general to restrict the partners referred to by relations, as has been discussed for XML topic maps. OO interfaces permit relations to be applied to different sets of concepts.

There is no separation between relation types and instantiated relations. Thus, it is possible to represent a kind of relation taxonomy, but it is not supported by the language. Also, not every relation has to possess a type. Although this fact is a degree of freedom regarding the language's expressiveness, it reduces automated processability. As no examples were found during this research work, demonstrating the need for this feature in product development, it is considered more beneficial to introduce a dedicated meta-layer for relation types to raise reasoning efficiency and improve semantic clearness. As for RDF-S, relations are invariably binary. This is a real drawback for the representational clearness in the product development domain, as has been argued in the section *Resource Description Framework (Schema), RDF(-S)* above. The same holds true for the fact that relations cannot relate to other relations.

The logic-like distributed and incremental representation of concepts has already been judged poorly suited for this work's targets for reasons of reduced maintainability and efficiency of retrieval. The same is true for set-oriented class definitions that potentially cover more than one class in a single statement.

Although OWL's expressivity is comparatively high due to its logic-like character, its potential to **map ontologies** onto each other is rather restricted, as there are solely three relation types for expressing equivalence and deviation. As OWL relations are binary by nature, it is not possible to express more complex relationships naturally. In product development (at least), concepts or instances are hardly ever really equivalent. They are typically similar in the best case. The kind of similarity, however, cannot be expressed by standard relationships and is also contingent on the viewpoint. As a consequence, it is not considered sensible to express some kind of standard equivalence or similarity of concepts and instances, but to express viewpoint-dependent, potentially very complex and n-ary relationships, instead. In other words, a simple equivalence-based ontology mapping is considered insufficient in the context of this work.

OWL supports **ontology versioning** by means of entering a version statement into a knowledge base. However, this works on the level of entire ontologies and not on individual entities. While the ontology-level versioning is certainly useful in dedicated cases, the more flexible approach would be to apply versioning information also to individual entities within the ontologies. This additionally reduces the amount of redundant information.

Summary. OWL's proposed use XML data types will be considered in this research's solution. Due to OWL's wide representational focus, in some fields it offers expressive power that is not needed in product development and which, in turn, is harmful for processing efficiency and manageability of information bases. A strictly encapsulated formalism is to be preferred within this research, as already motivated. In other respects such as the maximum arity of relations, meta-information on relations, and addressing of informational entities, OWL shows expressive deficiencies.

XMI and UML as Part of OMG's Four-Layer Meta-Model Architecture

Focus. In *[Jeckle, 2004]*, Mario Jeckle describes a four-layer meta-model architecture standardized by the Object Management Group (OMG)[*]. It comprises (1) the use of the *Unified Modeling Language (UML)*, for information modeling, (2) the use of *Meta Object Facilities (MOF)* for meta-modeling, and (3) the use of the *XML Metadata interchange format (XMI)* based on (4) *XML* for the stream-based interchange of models. These layers are depicted in Figure 12 below.

XMI is a text-based language for the complete description of arbitrary UML models, which are graphic-based, as is well known. It is designed to serve as an interchange format for these models, and thus complements MOF's collection of *IDL*[†] interfaces for the management of distributed meta-objects. The question here is whether it is suitable to be utilized as a GIS representation format.

☞ The term meta-data in this context denotes what is called abstract information in this thesis. It thus comprises concepts and relations serving as user information and as appropriate meta-information on several levels of abstraction.

The XMI standard comprises two major components:

~ ***XML DTD schema production rules*** specify how to produce (a) *XML document type definitions (DTDs)* or (b) *XML schema definitions (XSDs)* geared for the representation of individual meta-models represented in XMI.

~ ***XML document production rules*** set out the rules for encoding meta-information into an XML-compatible format. For instance, the XMI standard defines an XML vocabulary able to represent UML-based models.

Mario Jeckle states that most CASE and information model drawing tools such as *Rational Rose/XDE*™ or *Together*™ support XMI for exporting and importing UML models.

Footnotes

[*] See the URL http://www.omg.org.

[†] CORBA Interface Definition Language

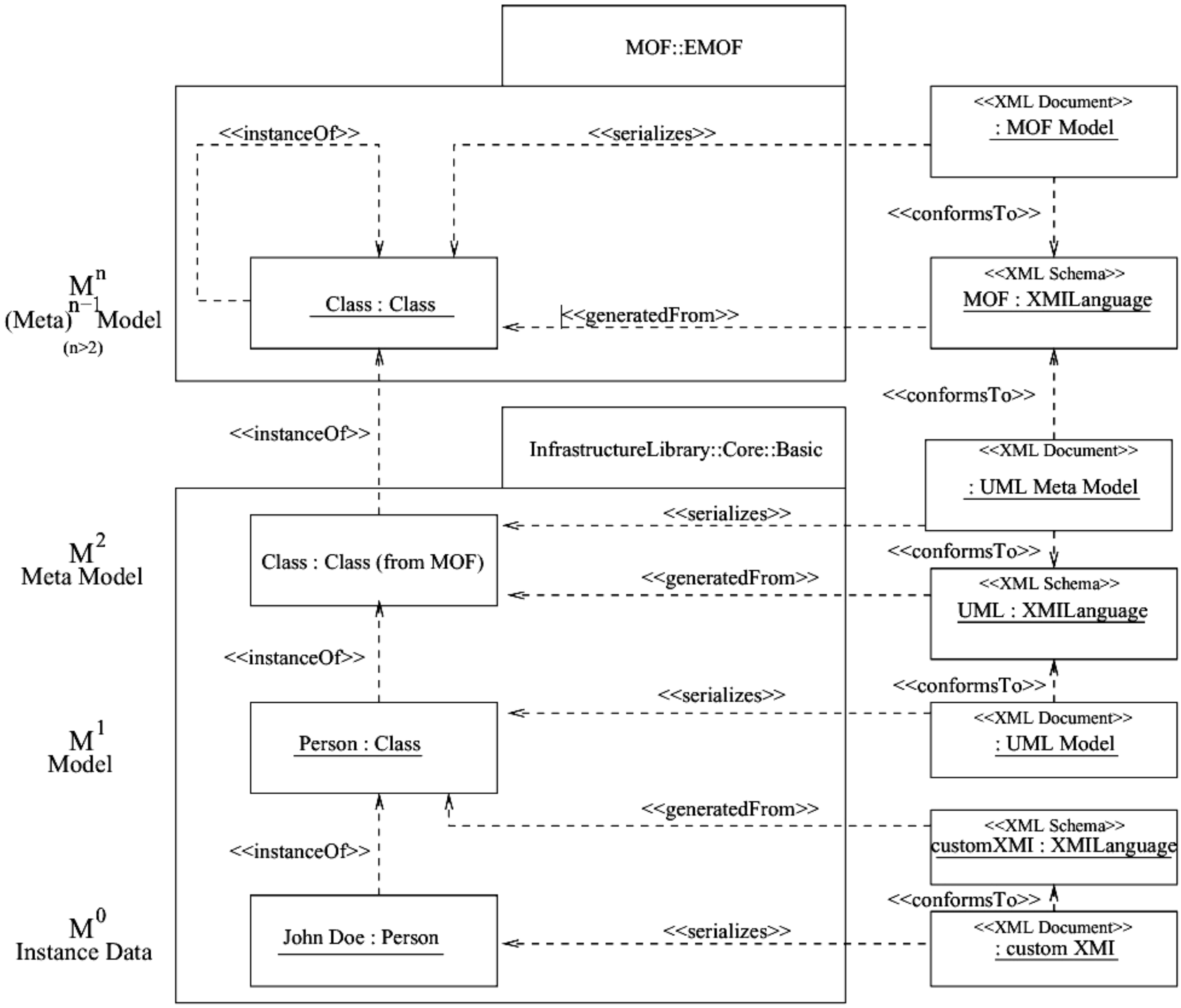


Figure 12: OMG's Four-Layer Meta-Model Architecture [Jeckle, 2004]

Information structures. XMI (and this is also true for MOF) knows concepts (*classes*), attributes, and relations (*associations*). *Classes* contain *attributes*, *associations*, and *compositions*. They may contain inherited elements as well.

XMI's solution of representing relations inside concepts originated in its function as an interchange format. For representation formats also used for information processing, this solution must be considered unsatisfactory, as (1) relations are not managed as individual entities of their own and (2) their representation is highly redundant, as each relation is duplicated within multiple concepts. This is a significant drawback from the perspective of this research; also, most other SCI approaches separate relations from concepts. Concepts should not have to be aware of their relations, as has been motivated in the section *What is needed? Deriving Requirements for Future Product Development IT Tools*. Unclear from the description is the question of whether relations are invariably binary. As XML schemas support single inheritance between concepts (*types*) only, while MOF and UML support multiple inheritance, XMI must copy inherited concept elements into the child concepts, which again produces redundancy. This fact urgently suggests that XMI schemas not be used as a representation format in the context of this research, since a major benefit of inheritance is lost and redundancy is generated. It even suggests discussing the usability of XMI to exchange general information at all (this is not a problem for specific information, as instances always carry all the attributes from all parent classes).

While OMG's four-layer meta-model architecture foresees a vertical model clustering according to abstraction levels, it obviously does not for horizontal clustering of models, e.g., achievable through contexts.

XMI offers a loophole, called *extensions*, to represent information that is not well expressible through its regular elements. However, this is a non-standardized loophole and thus not advantageous for placing the contents of a common representation language.

Addressing schema. The XMI examples, given by the OMG seem to presuppose the usage of unique identifiers for concepts, while attributes are addressed in a global scope using XML namespaces. However, even the potential usability of RDF's *URIrefs* would not meet the requirements for the targets addressed (see also the section *Resource Description Framework (Schema), RDF(-S)*). What is needed here is a unique addressing of informational entities within a global information space.

Summary. Although OMG's meta-model architecture seems to be a step in the right direction toward bringing order into the world of abstract models, it seems to have a crucial weakness: its concept and representation of relationships is too weak and relationships are not given the significance they need in the context of this research and also in many ontology-based approaches. A consequence is that information models using sophisticated relations have to be widely rearranged and encoded when converted into an XMI representation. Exchanging abstract models via XML is certainly a viable approach and is widespread in the meantime, as the discussion of other approaches has already showed. There, XML is also employed as a representation formalism for internal information storage and processing. However, in the context of this research, this seems to be appropriate for XML user data files only and <u>not</u> for XML <u>schemas</u>, as has been argued above. Therefore, XMI will not be adopted as GIS representation format in this work. As to the applicability of MOF and UML as GIS representation formalisms in this research, MOF shows the same weaknesses as XMI; UML is not streamable, as it is a pure graphical language.

Due to the deficiencies determined, this research will also not adopt XMI as a pure <u>interchange</u> format for a different GIS representation formalism.

7.1.5 Other Ontology-based Approaches

The Semantic Web's OWL is not the only ontology definition language. To give the reader a good impression of current solutions, several others are discussed below. They have been selected according to their degree of prominence in terms of available publications. There is also a number of other ontology engineering tools such as WebOnto, ODE, or KADS22 that are <u>not</u> discussed in this thesis. For a more tool-centric comparison, please refer to [Duineveld et al., 2004].

KAON

The Karlsruhe Ontology and Semantic Web Tool Suite (KAON) "was started in August 2001 as an internal research project of the knowledge management groups of the *Forschungszentrum Informatik (FZI)* and the University of Karlsruhe (TH)". *KAON is interesting in this work's environment, as it considers aspects of scalability and complexity of the representation formats and because it has achieved some practical implementations in the meantime.*

Focus. KAON is a combination of an ontology definition language with a collection of software tools. It targets various ontology-based application fields, including the Semantic Web (see *[Bozsak et al, 2002]*) and tries to avoid the complexity of OWL. The KAON language is descended from a "*simple core language*", called *Karlsruhe perspective on ontologies*, which, in turn, is based on RDF (see the discussion above and the appendix for an introduction). The **basic information types** supported are concepts and relations between concepts as well as partial orders on both and instances of them. Furthermore, explicit lexicalizations of concepts and relations are supported. Also, logical axioms are considered in the language's specification. The current version of KAON covers what the inventors call *ontology-instance (OI)* models on top of what has been described so far "*to support frequently* *occurring patterns of logical axioms such as the specification of algebraic characteristics of relations, e.g. symmetry, inverse, transitivity, and constraints on the cardinality of relations*". Basically, what has been said about RDF(-S) is also valid for the KAON language, e.g., regarding relations. Noteworthy is the attempt to avoid the often **unnecessary complexity** of highly expressive representation languages. The KAON developers state that, in many of their applications, the expressivity of OWL is not necessary, especially in terms of complex logical axioms. Consequently, they support OWL in parallel to their own KAON language. However, in the context of this research, all logic-based languages have been judged to be unnecessarily expressive in some respects. Therefore, the above-mentioned meta-information on relations will also not be expressed using logical axioms. It is also worth mentioning that instance information is handled separately from abstract (i.e., ontological) information.

Information retrieval. The KAON developer's guide *[KAON developer's guide, 2004]* discusses the interesting topic of how to query an ontology best. Although the KAON developers point out that they are in a trial phase, yet, they are trying to gain experiences with the implemented solution. They state the following querying principles applied in KAON: "*1. The ontologies are graphs of interconnected objects and the most appropriate way to explore them is through **navigation**. Hence, navigation is an intrinsic component of the query. 2. The ontologies consist of concepts (sets of objects) and properties (sets of object pairs). Since queries are applied on models with above mentioned structure, they should result in the model of the same structure. This means that the role of the queries is to define new (intentional) concepts and new (intentional) properties. The word intentional means that the content of such concepts and properties is not specified explicitly, by classifying instances and property instances one-by-one. Rather, the extension of the concept or a property is specified intentionally, by defining necessary conditions for memberships of elements in the concept or the property.*" These ideas basically match the ones developed in this research and motivated in the section *Further Theses*.

Architecture. KAON does not tackle the integration of applications. However, there is a collection of KAON tools available via APIs to be linked into productive applications, and Daniel Oberle (see *[Oberle, 2004])* is studying the applicability of a server-based architecture (**KAON server**) for Semantic Web applications. Such centralized, server-based solutions yield several benefits also for the informational integration of applications in the product development field. For example, applications may share a common, up-to-date information model hosting abstract information. Also, individual applications do not need to know about the existence or availability of other applications, if the server is able to abstract from this and to transparently find out information sources and destinations on its own. Also, application-spanning automation can be carried out much more powerfully. As has been stated above, the potential utilization of deductive database technologies for hosting information models is beyond the scope of this thesis and is subject to future work.

Summary. Especially interesting for this research is the fact that KAON developers also try to reduce the expressive power of their representation formalism. The same is true for the aspect of how to query a conceptual model.

Protégé-2000 and OKBC

Mainly due to its spreading, this section discusses another product of the Stanford University additionally to Ontolingua, called Protégé.

Focus. Protégé-2000[*] is basically a tool that "*allows the user to construct a domain ontology, to customize data entry forms and to enter data*" (see*[Protege, 2004]*). It was developed by Stanford

Footnotes

[*] See the URL *http://protege.stanford.edu/.*

Medical Informatics at the Stanford University School of Medicine with support from several other agencies. A good overview of features and applications is given on the Internet. Natalya Fridman Noy goes into some details of Protégé's *knowledge model* (see *[Fridman Noy et al., 2000]*) which claims to be completely compatible with OKBC, the *Open Knowledge base Connectivity Protocol* (see *[Chaudri et al., 1998]*) in order to achieve interoperability with other knowledge representation systems. The ultimate motivation behind this attempt is to achieve knowledge sharing and reuse. The OKBC API enables frame-based "knowledge" (i.e., information) models to be constructed and maintained. Fridman Noy lists *Ontolingua* and *Loom* as other compatible systems. She further explains that "*to function effectively as an access layer for many different knowledge-representation systems, it was important for OKBC to have an* extremely general *knowledge model. The set of representational commitments in the OKBC knowledge model is minimal and OKBC allows knowledge-representation systems to define their own behavior for many aspects of the knowledge model (e.g., default slot values)*". Consequently, Protégé restricts this "*extremely general knowledge model*" of OKBC in order to enable an efficient knowledge acquisition. It is worthwhile to record the fact that there is a tradeoff between interoperability and usability of knowledge-based systems, which is also recognized by Fridman Noy and her co-authors. Increasing generality of a representation formalism equals to decreasing semantic content. It has already been argued that a representation formalism for implementing a global information space between product development applications should be as specific (and thus semantic-loaded) as possible, thus using the chance of a rather restricted application field to maximize the support for automated information processing and sharing. This specialization has to stop at a point where it would restrict new information contents to be manageable. Although this point is not easily directly specifiable, it seems possible to exclude certain incarnations of especially high expressive power from being adopted. OKBC seems to show several of such features such as the fact that a frame can be a class, a slot and a facet at the same time. Also Fridman Noy maintains that "*this approach – allowing as many features as possible, requiring as few features as possible, and leaving some features underspecified – is perhaps the best approach for a common-access protocol, but designers of individual systems must make some design choices that restrict this generality. …We had to sacrifice some of the generality of OKBC to maintain easy-to-use and configurable knowledge-acquisition interface*"; please note that Protégé is a solution dedicated for **universal usability**. For product development, it is considered not necessary in this research to represent knowledge about the whole world and it is well possible to give specific guidelines for representing specific kinds of information, for sake of a formalism's semantic content and machine-processability. As a consequence, the use of OKBC is not considered to be adequate here.

A software plug-in for Protégé* allows management of OWL and RDF information within the system. This implementation was possible, thanks to the similarities in information handling and the openness of the software solution. It widens up Protégé's potential application fields significantly.

Information handling. Protégé is frame-based; however, its ontologies consist of *classes*, *slots*, *facets*, *axioms*, and *instances* of classes; this is purely declarative abstract and specific information, equivalent to abstract and instantiated concepts and relations between them. Classes are organized in a taxonomic hierarchy.

Attributes of classes and instances, as well as relations between them, are represented by slots. However, classes and instances and slots are implemented using frames. This means that relations (the *slots*) are represented more sophisticated as they are in pure frame-based approaches; they can be equipped with attributes, too (thus, Protégé should rather be classified as a modified frame-based approach). Furthermore, there is no hard distinction between classes and instances: in Protégé-2000, both individuals and classes themselves can be instances of classes. On this basis, *meta-classes* are introduced (see below): they are classes whose instances are themselves classes. They are supposed to reduce logical errors in the acquired information.

Footnotes

[*] See the URL *http://protege.stanford.edu/plugins/owl/index.html* for the *Protégé OWL plug-in* site.

Although slots can be equipped with attributes, they cannot refer to other slots, i.e., relations cannot refer to other relations. Slots are always binary, globally defined (independently from frames) and consecutively attached to them. *Template slots* can be attached to class frames only. A template slot attached to a class is inherited by its subclasses. Furthermore, a template slot on a class becomes an *own slot* of the instances of that class. An *own slot* attached to a frame describes properties of an object represented by that frame (an individual or a class). *Own slots* attached to a class do not get inherited to its subclasses or propagated to its instances. All slots are individual instances of frames and therefore cannot form a subclass/superclass hierarchy.

Slot *facets* allow the specification of meta-information on them such as the cardinality and legal "values" (other frames). Generally speaking, they define restrictions on an attachment of a slot to a class frame. There is multiple inheritance between classes.

The system provides reasoning that considers the inheritance of slots.

As has already been argued above, it seems useful from this research's viewpoint to equip relations with attributes. However, typing of and inheritance between relations are missed in the Protégé approach. Some other significant drawbacks are frame-system-typical: the impossibility to refer to relations within relations and the lack in terms of encapsulation: a global definition of properties is to be considered sub-optimal here, as has already been argued above.

Handling of application-spanning information and facilitation of communication is not directly supported by the system.

Meta-classes support **information acquisition**. Meta-classes are used to create corsets for information to be acquired and which is insertable into an information base (ontology). It is kind of a philosophical discussion, whether to implement a meta-level above the very information model (ontology) for purposes of restricting and controlling information input. However, information input is considered a separate task from information storage and management, which can be managed by the use of regular, i.e. non-meta classes and relations as well and which is consequently, for reasons of generality, not to be considered within the representation formalism. Furthermore, looking at the goal of a global information space, it is not considered to be a centralizable duty to care for correct information acquisition. One reason is the need for continuous maintenance of the meta-level before new information in enterable, which would contradict the idea of a generally unrestricted information sharing. However, the concept of using meta-level information for the control of information acquisition is well worth being considered for the decentrally running information processing of individual productive or service applications.

Summary. Although the adoption of the OKBC protocol for the purposes of this research has been declined above, some of Protégé's properties are worth being noted. This is especially true for the idea to (1) implement classes and slots on top of frames and thus enhancing the importance of relations and to (2) control the information acquisition using model contents, although the implementation in detail has been stated to be arguable.

7.2 Group 2 – Standardization Approaches To Application-neutral Data Exchange

This section discusses sub group 2a in more detail (exchange formats)*, while the means of* inter-process communication *(sub group 2b) are well considered, but excluded from a closer discussion, as they are standard technologies. Sub group 2a comprises the most prominent exchange format for CAx information, namely* STEP/ISO10303, *but also domain-neutral exchange formats based on* XML. *In this section only STEP and derivates are discussed as XML is already a commonly known formalism, and as it has already been included in the discussion of the Semantic Web approaches. There, several special XML formats have been discussed, and certain commercially available formats such as* iXF[*] *are considered too specific and rigid in expressive power for being included into this survey. The* Knowledge Interchange Format (KIF) *has already been considered in the discussion of Ostermayer's* Pragmatic-Situative Knowledge Representation. *Consequently, the remainder of this section is dedicated to sub group 2a* – Standardization Efforts For CAx Data Exchange.

It has been stated above that neutral (i.e., non-proprietary) information flow between applications is desirable because it is a prerequisite for a global information space that is open for applications from any software vendor. For this purpose, neutral representation formats must not be specialized to certain domains in product engineering but have to be applicable universally. Furthermore, it has been argued above that this can be reached by assuring high, but balanced expressiveness and the ability to transport context-related semantics.

7.2.1 STEP – ISO 10303

Focus. The *Standard for the Exchange of Product Model Data* (*STEP*), is an international standard (ISO 10303) aiming at the neutral and formal description of all product-defining data along the entire product life cycle. As the STEP definitions are implementation-independent, they allow (1) a file-based exchange of product data, as well as (2) the implementation and use of shared databases and (3) the archiving of product data (see *[ISO, 1994]* and *[Grabowski et al., 1994]*). Additionally, STEP targets to be utilized for application-internal information processing. However, today, STEP is primarily used for file-based data exchange.

Architecture. In the center of the STEP approach is an **integrated product model** (inter-mediate model). This is formed by about 1500 individual *integrated resources* as defined in ISO 10303 (see *[Ostermayer, 2001]* and Figure 13 for an overview). Ostermayer further states that thus far, the partial models of STEP's product model are oriented towards functionality of today's CAx systems, i.e., they mainly focus on geometry and topology, product structure, tolerances, FEM, and similar issues. Much other product information is still left to be detailed, however.

The STEP inventors did not intend that anyone should implement the whole standard. They rather recommend defining **views** on the integrated product model suitable for representing information from dedicated application domains. Such views are called *application protocols* (*APs*), in STEP (see Figure 13 for a list of important application protocols). For example, application protocol AP 214 describes core data for automotive mechanical design processes (see Table 1), AP 224 covers product and process planning, and AP 203 covers the product description without features. ISO 14649, also called *STEP-NC (STEP-Compliant Data Interface for Numerical Controls)*, handles the manufacturing process utilizing machining features. Application protocols may refer to more than one integrated resource. For instance, AP 214 refers to the integrated resource defined in STEP part 42 (geometry and topology), and to part 47 (tolerances).

Footnotes

[*] By Dassault Systèmes

AP 201	Explicit Draughting
AP 202	Associative Draughting
AP 203	Configuration Controlled 3D Designs of Mechanical Parts and Assemblies
AP 204	Mechanical Design Using Boundary Representation
AP 210	Electronic Assembly, Interconnect and Packaging Design
AP 212	Electrotechnical Design and Installation
AP 214	Core Data for Automotive Mechanical Design Processes
AP 215	Ship Arrangement
AP 216	Ship Molded Forms
AP 218	Ship Structures
AP 220	Process planning, manufacturing assembly of layered electrical products
AP 221	Functional data and their schema representation for process plants
AP 223	Exchange of design and manufacturing product information for cast parts
AP 224	Mechanical parts definition for process planning using machining features
AP 225	Building elements using explicit shape representation
AP 226	Ship Mechanical Systems
AP 227	Plant spatial configuration
AP 232	Technical data packaging: core information and exchange

Table 1: The Most Important STEP Application Protocols [ProSTEP iViP*]

In order to **develop** a STEP **application protocol**, it is necessary to follow a certain methodology:

1. Specification of a domain-specific *Application Activity Model (AAM)* describing the product development activities to be considered in the later model.
2. From the AAM, an *Application Reference Model (ARM)* is to be derived reflecting the user's viewpoint and describing requirements for the information to be represented.
3. The last step is to map the ARM contents onto STEP's integrated product model (resources). This mapping is described inside a static information model called *Application Interpreted Model (AIM*).

In other words, the resulting AIMs are standardized product models for specific domain in product development. They represent the respective domains' view on the integrated product model, and are linked to it by mapping relations. Consequently, AIMs are the basis for the development of so-called *STEP processors*, i.e., applications able to perform conversion between other information models and the model of a dedicated AIM.

✎ AIMs are also usable for the specification of database schemas when realizing persistent information storage. Part 21 of the STEP standard (see bottom of Figure 13) defines mapping rules describing how to produce STEP exchange files from an AIM; the STEP parts 22, 23, 24 and 26 define how to create SDAI structures and functions, C++ classes, C structures, or CORBA/IDL object definitions, respectively.

Footnotes

[*] See the URL *http://www.prostep.org/en/standards/was/ap/uebersicht/*.

The above-described theoretical solutions have not infiltrated practical product development: instead of a common, integrated product model only combinations of application protocols plus bi-directional converters are commonly applied. These are peer-to-peer solutions not providing any real integration over multiple applications. Thus, a global information space has not been achieved.

The ***ProSTEP iViP*** organization[*] has developed a CASE tool for developing STEP-based software applications and provides a graphical modeling language called *EXPRESS-G*. Graphical EXPRESS-G models are translated into schemas of text-based ***EXPRESS*** code. Using the methods described above, the CASE tool generates code for programming languages as well as database schemas.

STEP can be classified as being an **intermediate-model** approach. Such solutions inherently suffer from constantly limping behind the abilities of the systems to be integrated (see also section *7.4* on intermediate-model-based feature linking approaches). The sets of classes and relations offered by individual application protocols tend to require updates each time a software application offers or requests additional information; this may happen, for instance, on the implementation of new functionality or the provision of new user-defined feature types. As a consequence, information sharing is always restricted to the expressive power of the central model.

STEP does not offer means for implementing online communication or inter-application automation.

Information structures. STEP's object-oriented representation formalism EXPRESS is used to represent the integrated resources and the implementation-neutral models (AAM, ARM and AIM) described above.

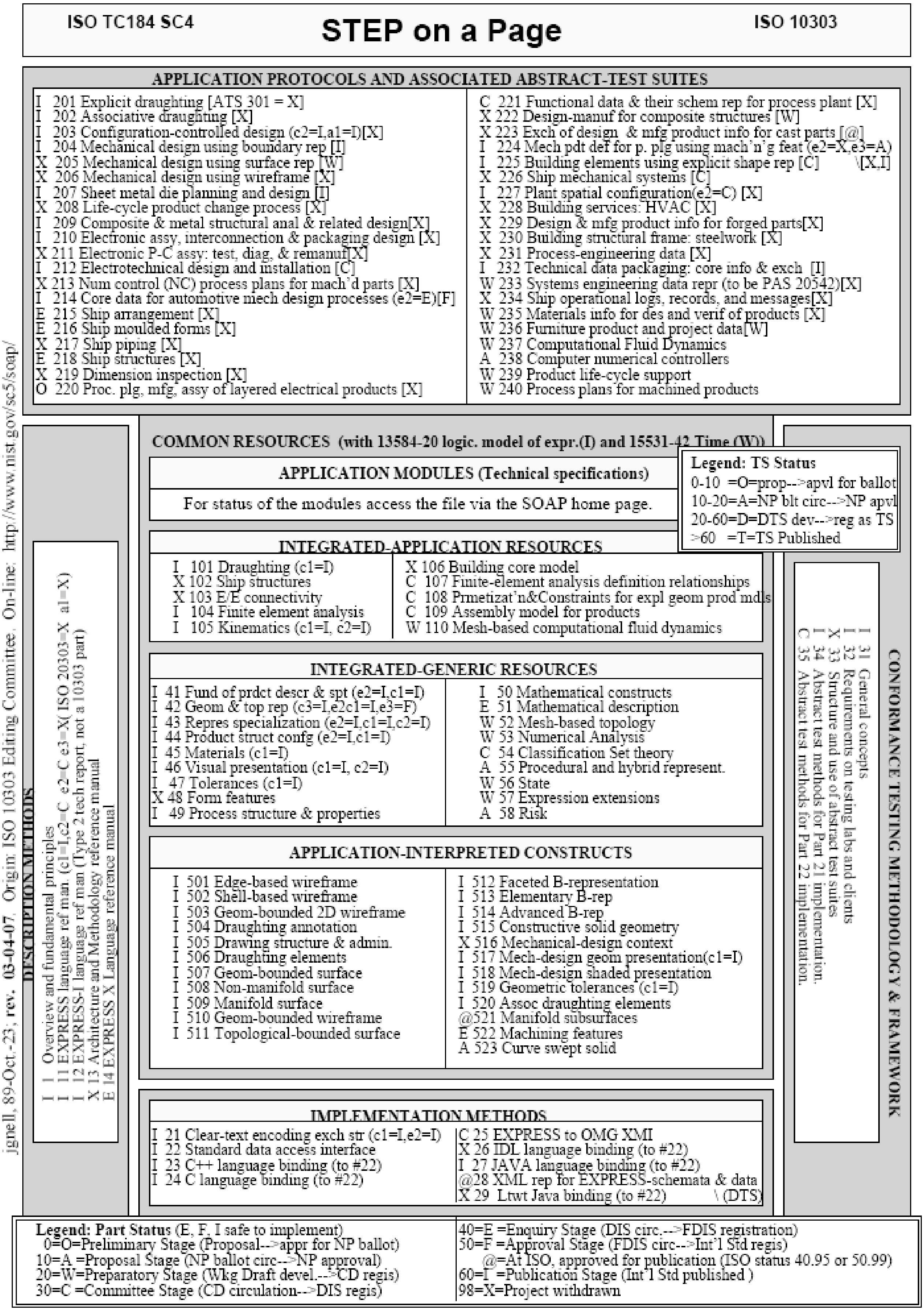
To describe static information, EXPRESS provides the following representational elements: concepts (called *entities*), constants, inheritance relations, aggregation relations, simple and user-defined data types including enumerations and declarative rules. Functions and procedures allow the representation of procedural information. *Schemas* allow the clustering of EXPRESS models and their referencing by others.

Although EXPRESS offers the option to represent procedural information, which is relevant for covering engineering strategies of different kinds (see section Part II – 3.2), its **expressiveness** shows significant leaks especially regarding relationships. As already mentioned, it is crucial for the integration of applications that a representation formalism considers relations as independent entities equipped with attributes and optionally with methods; furthermore, they are to be arranged in a meta-taxonomy. Additionally, formalisms should provide appropriate means for addressing informational entities all over a global information space.

Summary. Due to its intermediate-model philosophy trying to standardize the heterogeneous world's terminology, STEP is not suited to adequately integrate changing and evolving applications. Instead, approaches not using intermediate models are considered to be more powerful and flexible in exchanging information. Recommendable future enhancements of EXPRESS's expressiveness have been identified.

Footnotes

[*] See the URL *http://www.prostep.org/en/*.



ISO TC184 SC4 **STEP on a Page** ISO 10303

APPLICATION PROTOCOLS AND ASSOCIATED ABSTRACT-TEST SUITES

I 201 Explicit draughting [ATS 301 = X]
I 202 Associative draughting [X]
I 203 Configuration-controlled design (c2=I,a1=I)[X]
I 204 Mechanical design using boundary rep [I]
X 205 Mechanical design using surface rep [W]
X 206 Mechanical design using wireframe [X]
I 207 Sheet metal die planning and design [I]
X 208 Life-cycle product change process [X]
I 209 Composite & metal structural anal & related design[X]
I 210 Electronic assy, interconnection & packaging design [X]
X 211 Electronic P-C assy: test, diag, & remanuf[X]
I 212 Electrotechnical design and installation [C]
X 213 Num control (NC) process plans for mach'd parts [X]
I 214 Core data for automotive mech design processes (e2=E)[F]
E 215 Ship arrangement [X]
E 216 Ship moulded forms [X]
X 217 Ship piping [X]
E 218 Ship structures [X]
X 219 Dimension inspection [X]
O 220 Proc. plg, mfg, assy of layered electrical products [X]

C 221 Functional data & their schem rep for process plant [X]
X 222 Design-manuf for composite structures [W]
X 223 Exch of design & mfg product info for cast parts [@]
I 224 Mech pdt def for p. plg using mach'n'g feat (e2=X,e3=A)
I 225 Building elements using explicit shape rep [C] \[X,I]
X 226 Ship mechanical systems [C]
I 227 Plant spatial configuration(e2=C) [X]
X 228 Building services: HVAC [X]
X 229 Design & mfg product info for forged parts[X]
X 230 Building structural frame: steelwork [X]
X 231 Process-engineering data [X]
I 232 Technical data packaging: core info & exch [I]
W 233 Systems engineering data repr (to be PAS 20542)[X]
X 234 Ship operational logs, records, and messages[X]
W 235 Materials info for des and verif of products [X]
W 236 Furniture product and project data[W]
W 237 Computational Fluid Dynamics
A 238 Computer numerical controllers
W 239 Product life-cycle support
W 240 Process plans for machined products

DESCRIPTION METHODS

I 1 Overview and fundamental principles
I 11 EXPRESS language ref man. (c1=I,c2=C e2=C e3=X(ISO 20303=X a1=X)
I 12 EXPRESS-I language ref man (Type 2 tech report, not a 10303 part)
X 13 Architecture and Methodology reference manual
E 14 EXPRESS X Language reference manual

COMMON RESOURCES (with 13584-20 logic. model of expr.(I) and 15531-42 Time (W))

APPLICATION MODULES (Technical specifications)

For status of the modules access the file via the SOAP home page.

Legend: TS Status
0-10 =O=prop-->apvl for ballot
10-20=A=NP blt circ-->NP apvl
20-60=D=DTS dev-->reg as TS
>60 =T=TS Published

INTEGRATED-APPLICATION RESOURCES

I 101 Draughting (c1=I)
X 102 Ship structures
X 103 E/E connectivity
I 104 Finite element analysis
I 105 Kinematics (c1=I, c2=I)

X 106 Building core model
C 107 Finite-element analysis definition relationships
C 108 Prmetizat'n&Constraints for expl geom prod mdls
C 109 Assembly model for products
W 110 Mesh-based computational fluid dynamics

INTEGRATED-GENERIC RESOURCES

I 41 Fund of prdct descr & spt (e2=I,c1=I)
I 42 Geom & top rep (c3=I,e2c1=I,e3=F)
I 43 Repres specialization (e2=I,c1=I,c2=I)
I 44 Product struct confg (e2=I,c1=I)
I 45 Materials (c1=I)
I 46 Visual presentation (c1=I, c2=I)
I 47 Tolerances (c1=I)
X 48 Form features
I 49 Process structure & properties

I 50 Mathematical constructs
E 51 Mathematical description
W 52 Mesh-based topology
W 53 Numerical Analysis
C 54 Classification Set theory
A 55 Procedural and hybrid represent.
W 56 State
W 57 Expression extensions
A 58 Risk

APPLICATION-INTERPRETED CONSTRUCTS

I 501 Edge-based wireframe
I 502 Shell-based wireframe
I 503 Geom-bounded 2D wireframe
I 504 Draughting annotation
I 505 Drawing structure & admin.
I 506 Draughting elements
I 507 Geom-bounded surface
I 508 Non-manifold surface
I 509 Manifold surface
I 510 Geom-bounded wireframe
I 511 Topological-bounded surface

I 512 Faceted B-representation
I 513 Elementary B-rep
I 514 Advanced B-rep
I 515 Constructive solid geometry
X 516 Mechanical-design context
I 517 Mech-design geom presentation(c1=I)
I 518 Mech-design shaded presentation
I 519 Geometric tolerances (c1=I)
I 520 Assoc draughting elements
@521 Manifold subsurfaces
E 522 Machining features
A 523 Curve swept solid

CONFORMANCE TESTING METHODOLOGY & FRAMEWORK

I 31 General concepts
I 32 Requirements on testing labs and clients
X 33 Structure and use of abstract test suites
I 34 Abstract test methods for Part 21 implementation.
C 35 Abstract test methods for Part 22 implementation.

IMPLEMENTATION METHODS

I 21 Clear-text encoding exch str (c1=I,e2=I)
I 22 Standard data access interface
I 23 C++ language binding (to #22)
I 24 C language binding (to #22)

C 25 EXPRESS to OMG XMI
X 26 IDL language binding (to #22)
I 27 JAVA language binding (to #22)
@28 XML rep for EXPRESS-schemata & data
X 29 Ltwt Java binding (to #22) \ (DTS)

Legend: Part Status (E, F, I safe to implement)
0=O=Preliminary Stage (Proposal-->appr for NP ballot)
10=A =Proposal Stage (NP ballot circ-->NP approval)
20=W=Preparatory Stage (Wkg Draft devel.-->CD regis)
30=C =Committee Stage (CD circulation-->DIS regis)

40=E =Enquiry Stage (DIS circ.-->FDIS registration)
50=F =Approval Stage (FDIS circ-->Int'l Std regis)
@=At ISO, approved for publication (ISO status 40.95 or 50.99)
60=I =Publication Stage (Int'l Std published)
98=X=Project withdrawn

jgnell, 89-Oct.-23; **rev. 03-04-07**, Origin: ISO 10303 Editing Committee, On-line: http://www.nist.gov/sc5/soap/

Figure 13: STEP on a Page [Nell, 2001]

7.2.2 Further STEP Activities

Several other activities base on the STEP standards. This section briefly discusses SUMM *being one of them. The running NIST program* Product Engineering *will be sketched in the section* Update on the State of the Art *below.*

Overview. *SUMM* is considered to be of relevance for this discussion since it tackles semantics of STEP models. Another, broader, approach, *PDTnet* (see *[PROSTEP, 2003]*), strives to achieve "online PDM web integration" and is part of OMG ManTIS's *PLM Services* standard (see *[OMG PLMservices, 2004]*). PDTnet develops platform-independent PLM services based on an (intermediate) reference model that is editable through UML tools (see Figure 14).

☞ As all projects mentioned rely on STEP's principle philosophy, they do not provide significant new insights to this research, in addition to the ones discussed above.

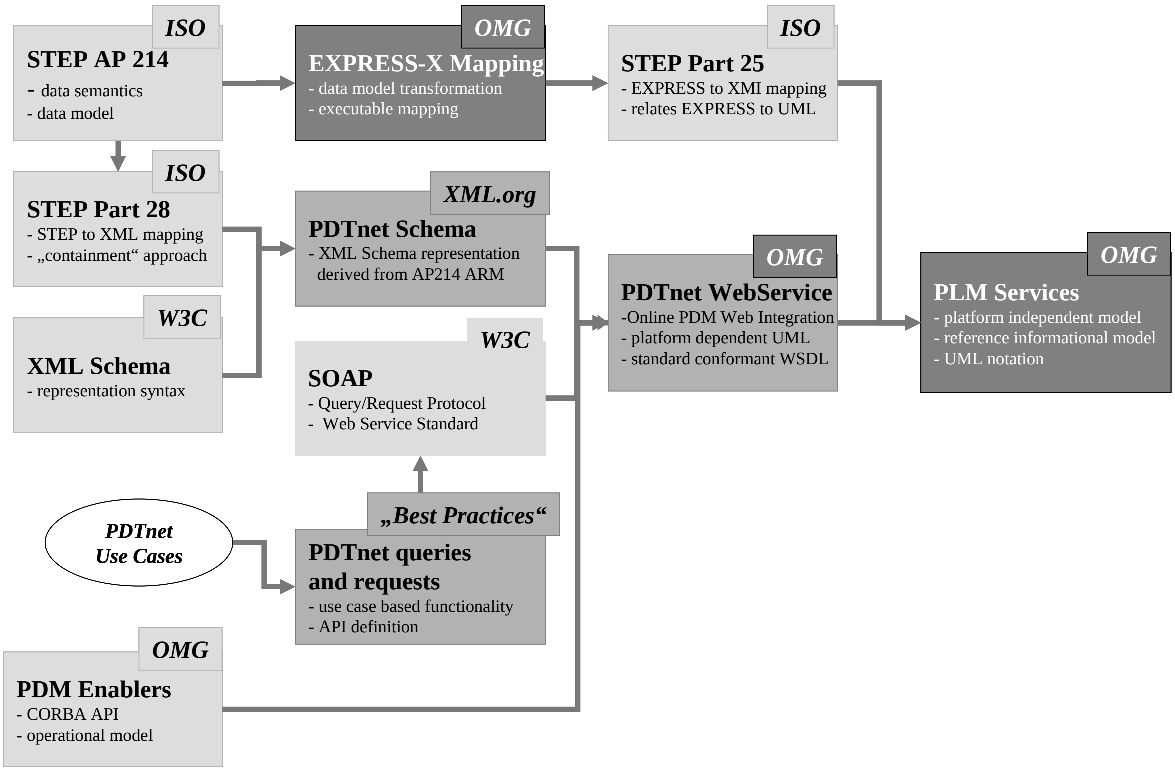


Figure 14: OMG's PLM Services and PDTnet [PROSTEP, 2003]

NIST: SUMM. The *American National Institute of Standards and Technology (NIST)* pushes activities in ISO workgroups for achieving interoperability of product models (see *[NELL, 1998]*). A milestone for reaching this goal was the development of the *ISO/IEC JTC1 Semantic Unified Meta Model (SUMM*, see *[ISO SUMM, 1991]*).

The NIST describes the role of SUMM as follows: *"The unified approach*[*] *assumes that there exists a common meta-level template across constituent models, providing a means for establishing*

Footnotes

[*] (to enterprise-model interoperability, remark of the author)

semantic equivalence. The metamodel is not an executable form as it is in integrated situations. Using the metamodel, any model can be translated into any other. Loss of some semantics is possible. Normalized semantics is established by owners of constituent models. The ISO/IEC JTC1 Semantic Unified Metamodel, SUMM, is an example of a unified-model template." Filippo Salustri adds in *[Salustri, 1996]* that *"SUMM (Fulton, 1992) attempts to provide an underlying logic based on the predicate calculus for the PDES/STEP product model initiative."*

Although SUMM adds the coverage of semantics to STEP models, it still adheres to STEP's philosophy of a static, standardized and non-executable central information model. Therefore, from the perspective of this research, it has to be judged to be less effective and flexible than a solution directly correlating the contents of applications' information models. Additionally, such integration is to be documented within a model that is interpretable and changeable at runtime. This has already been motivated in the discussion of STEP above. Furthermore, logic-based representations have been shown to be inadequate in the context of this research (see also section *Logics and Sub-symbolic Representation Formalisms* above).

7.3 Group 3 – Integration Architectures

The group of integration architectures is heterogeneous. Therefore, this section sets out several approaches that are largely unrelated.

Overview. In the group 3 of *integration architectures* several approaches have been investigated: the ***Engineering Portal (EP)*** is an OEM-internal solution whose concepts are already implemented and applied in the automotive production. The EP is a typical Information System in that it supports its users with instance information from several source systems. Ulrich Frank states that "*information systems typically contain representations of numerous instances. The conceptualization of instances of the same kind (through classes or types) happens usually outside the boundaries of the system that the user has access to*" (see *[Frank, 2002]*). In his paper, Frank suggests to integrate Knowledge-Management systems (KMS) and classical Information Systems. Although he puts some levels of abstract models on top of the IS meta-models, his solution still depends on an integration through common concepts, while the relations possess weak (abstract) semantics. For this reason, this approach will not be discussed more closely here. Nevertheless, the large field of (Corporate) Information Systems is considered relevant, in that interoperability with a new approach should be supported. This includes systems classifiable as Decision Support Systems (DSS), Management Support/information Systems, Expert Systems, or Workflow Managment Systems. ***High-level Architecture (HLA)*** is promoted to become a standard for IT systems involved in the military domain. HLA takes into account numerous practical issues including the inter-operability and robustness of systems. Classical **middleware** approaches contribute application-neutral issues from computer science regardless of any need for backward compatibility to legacy software. This provides more freedom in software-architectural design decisions. As middleware approaches are widely spread, they will be sketched only. The typical **ontology mapping** approach will also be investigated below, as it aims at the integration of abstract information, which is commonly not found in other recent approaches. Not discussed here are commercially available **integrated CAx and PLM solutions**, as they are not suitable to integrate third-party software and do not consider abstract information appropriately (i.e., in dedicated and adequately-structured models, see above). In addition, they do not process semantics of information. Their internal means of integrating PPR models are custom-made to special purposes such as the maintenance of a uni-directional associativity based on geometric dependencies between features, and are not open for external access via APIs. Relations applied are usually uni-directional and de-coupled from any abstract information. Also not discussed further is the

PDM enabler technology[*] (see *[OMG PDMenabler, 2000]* for an overview) of the *Object Management Group's (OMG) Manufacturing Technology and Industrial Systems Task Force (ManTIS)*[†], since it does not really provide insights not already gained from the discussion of other approaches in this survey of the state of the art. Nevertheless, a short quotation will be given here from *[OMG PDMenabler, 2000]*: *"The PDM Enablers are a standards-based Application Programming Interface (API), specified in IDL, that makes PDM services available in a CORBA environment to other systems that require them (such as CAD, CAE, CAM systems mentioned above, and even other PDM systems). Following the CORBA model, CAx systems can use the PDM Enablers API and the standard network interfaces to interact directly with any conforming PDM system. …As stated in other parts of this document, the OMG PDM Enablers and the STEP PDM Schema are complementary specifications that work together. They can be viewed as separate necessary documents that concentrate on different aspects of a single unified specification of a standard system of interoperable PDM clients and servers. The Reference Model for Open Distributed Processing (RM-ODP) provides a conceptual framework in which each of these specifications plays a separate well-defined role"*. Also the ManTIS' ***CADservices*** standard[‡] will not be discussed in more detail, although it provides a means of accessing CAD systems via a neutral, CORBA-based interface that allows the manipulation of CAD models. Although this is considered interesting from an automation viewpoint, the information model behind the interface is very restricted in terms of the set of supported concepts (i.e., the range of transferable EO instances) and especially in terms of the set of provided relations between them (see *[OMG CADservices, 2003]*). Furthermore, the interface does not at all handle abstract information. Management of contexts and addressing is missing or insufficient for being utilized within a GIS. The CADservices' set of manipulation services may serve as a basis, once pertinent services will become of relevance in this research's future work. However, this issue is not considered to be in the center of this research's targets.

7.3.1 Engineering Portal

The *Engineering Portal (EP)* offers to engineers a single-GUI[§] access to several heterogeneous *Engineering Data Management (EDM)* systems, e.g., hosting information on bill-of-materials, PDM information, or supplier parts. This information is specific information. The EP uses an intermediate class model on which it maps the EDM systems' individual models. Especially interesting is the fact that the central EP model relies on a simplified STEP AP214 data model. "Simplified" means "less classes, i.e., abstract concepts", since the full range proved to be too inefficient. Applications are integrated based on a *J2EE*[**] framework: a central, service-based mapping application stores the whole model within the RAM. CORBA and Web Services[††] are supported to connect to the EDM systems acting as servers. The EP reads the details on the services to be called on retrieval of the various model entities at runtime from a mapping table within a database. The GUI offers views that are configurable through XML files.

Brief discussion. The EP is a powerful system to grant engineers access to instance information. Nevertheless, its functionality differs significantly from what is desired in this research. Abstract information is not provided (not least, because the integrated systems don't offer it), and the integration does not dynamically handle semantics. This means that the mapping is done manually by

Footnotes

[*] See the workgroup at URL *http://mantis.omg.org/mfgppepdm.htm#PDMRTFV1.4*.

[†] See the URLs *http://mantis.omg.org/index.html* and *http://mantis.omg.org/mfgppe.htm*.

[‡] See the workgroup at URL *http://mantis.omg.org/mfgcadv1-2rtf.htm*.

[§] Graphical user interface

[**] Java 2 Platform, Enterprise Edition

[††] PDM web connectors

means of filling the mapping tables. This happens without changing the mapping applications' software code. The runtime interface used between EP and the connected EDM systems is not uniform as the set of functions to be called varies from adapted system to system. As there are no persistent links between the individual applications' EO instances, the mapping system has to resolve the mapping on each call based on the mapping tables and the EO instances' types and IDs. Searching mapping tables is time-consuming and therefore the set of different EO classes had to be restricted. Re-use of engineering solutions is practically improved by the fact of integration as such and by involving a dedicated system for this kind of information.

✍ Due to its organizational embedding, the EP integrates applications managing huge amounts of existing data without having available persistent links. In contrast, this research aims at integrating applications filling up their information pools from the scratch, once the integrative solution is implemented. This includes modifying the applications in terms of adding awareness of relations to other applications, as this is viewed to be a prerequisite of cooperation.

✍ Also existing and filled PPR models can be integrated by means of post-processing, which is costly if overall performed, however. A case-to-case integration of existing PPR models would be probably more adequate.

7.3.2 High-Level Architecture

With *high-level architecture (HLA)*[*], e.g., summarized in *[Kümmeth, 2004]*, informational integration happens based on a sort of inter-mediate model, called *data dictionary*, on whose entries (class attributes) all individual applications' information structures are statically mapped. Furthermore, numerous infrastructural means are offered through globally accessible services for supporting communication and cooperation of distributed applications. Special interaction classes allow having the system automatically execute algorithms (e.g., performing data transfers) when certain trigger conditions appear. There is no dynamic interpretation of semantics. For reasons of space, no more details, but only the **conclusions** will be given now shortly:

- Service-based provision of a global model assures to find the needed mapping information.
- Availability of global services reduces the implementation effort for client applications.
- The data dictionary is attribute-based; (rigid) semantics just for attributes → too low-level, no EO classes, no relations between them, no global information model, too many indirections such as decentral SOMS.
- The approach to map individual informational entities of applications improves information flow between them.
- HLA is costly to implement.
- HLA is very flexible as any constellation of systems may be added and removed from the running network.

Footnotes

[*] See the URL *https://www.dmso.mil/public/*.

7.3.3 Middleware Approaches

Classical centralized **middleware approaches** (see *[Emmerich, 2000]* for a high-level discussion) commonly share the following principles:

- A single, central middleware application offers services to productive applications (*front office systems*) while falling back on *back office systems* (mediation functionality)[*].
- A single, common inter-process communication interface and protocol is applied. The middleware application and each client application utilize this common interface.
- The architecture is centralized, in that it…
    - provides one central access point for calling global services,
    - allows the maintenance of a central directory for all kinds of information,
    - allows the middleware to maintain the central control of applications (control flow) for automation purposes.

This typically yields the following **benefits**:

- The communication interface is extendable (through additional services).
- The architecture is open to any kind of client application.

Conclusions. Due to its openness, flexibility and the support of automation, a **service-based approach** to communication between applications can be judged to be beneficial also for purposes of informational integration of applications and automation of tasks. Services allow the implementation of a global information space. Furthermore, Emmerich pleads for the utilization of middleware approaches in combination with **markup languages** in order to achieve semantic mapping between distributed applications (see *[Emmerich et al., 1999]*).

✍ For more even more flexibility, services that are offered by clients should be re-routed to other clients by the central server (middleware application), i.e., inserted into a central directory of services and accessed via the central server.

The **central server** is always accessible, as it is an invariant element of the architecture, and clients (ProSAps) do not need to know details about the existence of other clients. They merely are to be informed about the services available. This indicates the benefits of a central directory of services that is retrievable at runtime.

7.3.4 Approaches Utilizing Ontology Mapping

Ontology Mapping strives to logically integrate multiple ontologies. Pertinent representational elements in the OWL language have already been discussed. Subsequently, a quotation from *[Maedche et al., 2001]* is given, characterizing the typical motivation of such approaches: *"Ontologies as means for conceptualizing and structuring domain knowledge within a community of interest are seen as a key to realize the Semantic Web vision. However, the decentralized nature of the Web makes achieving this consensus across communities difficult, thus, hampering efficient knowledge sharing between them. In order to balance the autonomy of each community with the need for interoperability, mapping mechanisms between distributed ontologies in the Semantic Web are required. In this paper we present MAFRA, an interactive, incremental and dynamic framework for mapping distributed ontologies. We adopted a multi-strategy process (similar to [Doan et al., 2002]) that calculates similarities between ontology entities using different algorithms. …An ontology mapping process, as defined in [Rahm & Bernstein, 2001], is the set of activities required to*

Footnotes

[*] CORBA also falls into this category.

transform instances of a source ontology into instances of a target ontology”. MAFRA normalizes ontologies that are to be mapped by means of a uniform RDF(-S) representation in order to eliminate syntax differences …*“and making semantics differences between the source and the target ontology more apparent.”* A *similarity module* detects similarities between entities in the two ontologies to be mapped, based on several similarity measures from literature. Thus, MAFRA’s authors claim to support mapping discovery.

Brief discussion. Ontology mapping typically uses an inter-mediate model approach and maps multiple ontologies on this central inter-mediate model. The shortcomings of this method have already been mentioned and will be discussed in more detail with the feature linking approaches. Central issues are (1) the fact that inter-mediate models prohibit the existence of direct relations between informational entities and (2) that they limit the range of mappable information through their own terminology. Specific to MAFRA is the characteristic that n-to-m mapping relations are built-up of several binary relations and that they cannot relate to other relations (see the discussion of RDF(-S) above).

Conclusion. Considering the targets of this research, the typical ontology mapping approach is assessed to be interesting for assisting human experts in the task of integrating individual ontologies within a global information space; i.e., they are a means of supporting information acquisition in the broad sense, in that they can support the expert in the creation of relations, based on automatically detected similarities. However, for the given reasons, ontology mapping is not considered appropriate for directly integrating several models inside a global information space.

7.4 Group 4 – Approaches to Integration by Automation – Feature Linking

As motivated in the overview of the state of the art, this section discusses subgroup 4a, the approaches to feature linking.

Survey

It has already been brought up that domain-specific types of features and other Engineering Objects may lead to a logical separation of the product models hosting them.

☞ Current scientific approaches usually talk about “features”* instead of “objects” or “engineering objects”. Therefore, the further discussion within this section will also stick to this notion.

Scientific approaches covering ***feature mapping*** functionality – also called *feature* ***conversion*** or *feature* ***transformation*** systems – try to tackle this problem in various manners that are all based on the same underlying principle: a transformation between two feature sets A and B is performed by generating the new set of feature instances B from the given one A. In the general case there may be n-to-m relations between the feature instances of A and B. Pure feature mapping does not really close the gap between feature-based models, since it targets the generation aspect, neglecting what happens afterwards. It “jumps” the gap instead of closing it. To cover these new aspects, the concept of ***feature linking*** shall be introduced. Feature linking is informally defined here as feature mapping together with the generation and maintenance of persistent links between the mapped feature instances.

In the context of feature-based applications that are manipulating feature-based product models*, **automation** is equivalent to what feature mapping does: to generate parts of such models automatically by inserting feature instances into them.

Footnotes

* The notion of engineering objects and its given meaning has been introduced in this work.

Discussion

In the following, some comments will be given on several scientific approaches that are relevant to the field of feature linking.

Communalities

Which objects are considered? Current approaches focus on features and their constituents, describing the features' geometry such as vertices and planes.

Which building blocks are offered? In the approaches covered here, basic and compound features constitute the high-level[†] building blocks for constructing product models. Basic features may be aggregated to compound features.

Context and **semantics** are not covered. There is no **global information space**. Issues of **user assistance**, **scalability** and **flexibility** are not covered in the reviewed literature.

Characteristic: How Features Are Modeled

One key set of characteristics of feature-based systems describes the way in which the features are modeled. In this respect, most publications are rather vague, thus forcing the reader to make assumptions about most of the details of the models used inside the investigated approaches. It seems to have become standard in more recent approaches to **classify** features and to differentiate between feature **classes** and **instances** accordingly as well as to use **taxonomies**[‡] to arrange the feature classes. For instance, while Shah, Krause, and Srikantappa refrain from using taxonomies at all, De Kraker and Lecluse do use them (see *[Shah, 1988]*, *[Krause et al., 1991a+b]*, *[Srikantappa & Crawford, 1992]*, *[De Kraker, 1997]* and *[Lecluse, 1999]*). It is not clear, however, if any existing approach considers using one single (i.e., global) taxonomy for all known feature classes – at least this is not pointed out in any of the papers investigated by the author. Wong and Leung use separate taxonomies of feature classes for the individual feature-based applications (see *[Wong & Leung, 2000]*). As none of the systems considered provides this characteristic, they seem to get by without global taxonomies. Also, none of the investigated approaches models feature classes and classes of other engineering objects inside the same taxonomy.

Characteristic: How Automation (Mapping) Knowledge Is Modeled

Current approaches also differ in the way they model knowledge about mapping. Looking at where this information is stored, the literature yields several alternatives: (a) inside or (b) outside the feature definitions (classes). If outside, mapping knowledge could be represented (b1) in hard code inside the system's algorithm or (b2) stored inside a separate knowledge base. Another alternative – not found in the literature – would be (b3) to store this information within special relations that link the feature classes inside the class taxonomy.

✍ This solution could also be regarded as a special kind of knowledge base formed by a structured network tied to the feature classes as a second information layer. Any combination of the mentioned principle alternatives is also conceivable.

Storing mapping knowledge inside a feature class, alternative (a), means to have it partly where it belongs, and partly not, as mapping always occurs between at least two process steps. Another

[*] (Or, more general, PPR models)

[†] With an own engineering meaning

[‡] Hierarchy of feature classes based on inheritance relations

shortcoming is the need to change the feature classes themselves every time a destination ProSAp is added or removed from the process chain. Although these characteristics may appear to be not quite intuitive, solution (a) leads to a focused and clear representation, which enhances readability and maintenance. Krause, for example, sticks to this method, using a representation language called PDGL[*] to express the knowledge inside the feature classes (see *[Krause et al., 1991a+b]*). The most straightforward solution, method (b1), is partly employed by Shah, for example, where user-defined design features are mapped on generic machining features by means of a hard-coded CFT[†] reconstruction algorithm (see *[Shah et al, 1993]*). It is obvious that hard-coded solutions are generally faster in execution than interpreter-based ones, yet they lack flexibility and are more difficult to understand for the users of the system. Shah also uses approach (b2) for pre-defined design features. The system interprets production rules to map them onto machining process operations by choosing alternatives from a pre-defined machining process graph. The *PART*[‡] and *PART-S* process planning systems (see *[Van Houten, 1991]* for PART, and *[De Vin, 1994]* and *[De Vries, 1995]* for PART-S) store knowledge about design feature recognition within scripts, thus also following approach (b2). These algorithms are represented in a dedicated feature recognition language. Geelink developed a solution to automatically derive the algorithms from design feature definitions consisting of conceptual graphs (see *[Geelink, 1996]*). The algorithm automatically deriving the scripts for feature recognition is itself hard-coded (method b1). Han & Requicha use rules to weight mapping hints inside design features (see *[Han & Requicha, 1997]*). Wong & Leung convert neutral features into application-specific features using production rules (see *[Wong & Leung, 2000]*).

✍ Generally, rule-based approaches, just like any other **knowledge-based approach**, are very flexible in the sense that the mapping knowledge can be adjusted without changing the program, which can also be done by non-programmers. This is essential for practical use of a system since knowledge may change over time, and information about the handling of new feature types may have to be added. Additionally, because all available knowledge is stored within a single location, its consistency can be established quite easily. On the other hand, this property leads to comprehensibility problems, which make isolated knowledge bases – particularly large ones – hard to maintain.

Hybrid approaches are, for example, the hint-based systems, which store some of the information relevant for feature mapping directly inside the feature instances (a) and another part elsewhere, for instance hard-coded inside the program code (b1). Ishii and Miller, for example, use codes inside a design-feature-based model to derive downstream models (see *[Ishii & Miller, 1992]*). So far, the author's literature studies have not yielded any true examples for (b3)-like systems. **Conceptual graphs** (see *[Sowa, 1992]*), however, such as those used by Geelink for other purposes (feature definition and recognition) are an option for handling information about engineering objects and their interrelations – including mapping knowledge – in a clear fashion (see *[Geelink, 1996]*).

Direct or Indirect Mapping

There are systems such as Suh's and Wong's that convert feature-based models indirectly using an **intermediate model** that contains geometry-oriented[§], **application-neutral intermediate feature classes and instances** (see *[Suh & Wozny, 1998]* and *[Wong & Leung, 2000]*). Deviating from such

Footnotes

[*] Part Design Graph Language

[†] Constructive Feature Tree

[‡] PART is an acronym for Planning of Activities, Resources, and Technology

[§] The term *geometry-oriented* as used here means that every intermediate feature focuses on representing product geometry, although it may also contain other information.

neutral-model approaches, Bronsvoort and De Kraker use the design feature model as the central intermediate model (see *[Bronsvoort et al, 2001]* and *[De Kraker, 1997]*). Instead of performing real feature mapping, they use a **two-step feature recognition** approach to proceed from the design feature model to other, application-specific models. Yet, also this method allows changes inside the individual views and propagates them back to the intermediate model via **constraints that relate geometric entities**.

✍ The key advantage of **intermediate-model approaches** is that they reduce the number of necessary mappings between the feature-based applications. However, intermediate-systems also have approach-inherent drawbacks: they suffer from a certain lack of flexibility in that the set of expressible information is restricted to the expressiveness of the intermediate model. This model has to be able to represent all kinds of information that is relevant along the process chain. As a consequence, intermediate features may be considered all-purpose feature classes – in contrast to specialized and optimizable ones – which is cumbersome and leads to well-known effects of non-normalized data storage such as redundancy and partly inappropriate attributes. Another principle drawback of such approaches is that they **inhibit** expressing **direct relations** between features (or feature classes) of different applications. This is of importance not only for automation reasons, as especially the uncounted kinds of relationships between any kinds of Engineering Objects cannot be foreseen in a fixed terminology for general use.

Generation and Maintenance of Links between Mapped Feature Instances

Does a system record its mappings and does it create dedicated and persistent links of some kind between the respective objects? This aspect appears to be more or less out of the scope of current approaches, which tend to focus more on the automation aspects. Although it would be theoretically possible to provide for this, e.g., in the approach taken by Krause, there is no evidence that this has been done thus far (see *[Krause et al., 1991a+b]*). In the papers *[Suh & Wozny, 1998]* and *[Suh, 1995]*, which report about further examples of intermediate-model approaches, Suh claims to allow designers and application experts to communicate via their intermediate models. Yet, because direct links between ProSAp models are missing, which could bypass the intermediate model, this kind of communication appears to be restricted to features representing some kind of geometry (geometry-centered communication). It is not possible to directly link a feature to any other one or to any non-feature Engineering Object at all. In Bronsvoort's paper *[Bronsvoort et al, 2001]* the only approach is found that uses notions like "feature linking" and "inter-feature links", but their meaning also seems to be geometry-focused. The paper does not mention how the links are established.

Conclusions Drawn Regarding Feature-Linking Aspects

Existing commercial as well as scientific solutions seem to focus on automation aspects, seeing ProSAp integration more or less solely on the periphery – both in terms of the set of object types they are able to interconnect, as well as in terms of the kind of information the interconnections can convey. In terms of automation capabilities offered, most approaches tend to address specialized areas. There seems to be no system that is generally applicable for the whole range of objects relevant in engineering. In this sense, state-of-the-art systems do not yet close the informational gap between process steps to the degree desired by engineers in the context of this research.

How features are modeled: the capability to model all classes of features and other relevant Engineering objects inside a virtually single global information model is a prerequisite to being able to relate these entities to each other in a clear and straightforward fashion (see "direct relations") and to reason about them within the same overall context. A bit less formally put, one could say that these entities know about each other.

Characteristic: how automation (mapping) knowledge is modeled: The combination of methods for storage of knowledge about mapping within special relations (which inter-connect the feature classes inside the class taxonomy (b3)), with the use of conceptual graphs seems to avoid the aforementioned shortcomings of the systems set out above. The information should be stored in a highly targeted way, directly related to the objects it refers to. This makes it universally usable, flexible, intuitive, and easy to read and to maintain (edit, add, delete elements).

Direct or indirect Relations and Mapping: the ability to model relations between relevant objects and/or classes directly seems to be a prerequisite for sophisticated process chain integration. Hence, a solution that does not use intermediate models is proposed.

Generation and Maintenance of Links between Mapped Feature Instances: Links between sets of feature instances are the carriers of information for inter-ProSAp communication. In this respect, existing feature-based systems do not offer a solution that is universally applicable and that supports any kind of relationship.

Conclusions in Short:

~ A virtually single and global information model within the global information space is a prerequisite to locate all relevant information and to offer a clear and straightforward fashion of relating features and other entities to each other within the same overall context.

~ Storing mapping knowledge inside a kind of conceptual relations yields clear structures.

~ Since direct relations between models are required, intermediate models are not suitable.

~ There is no solution by existing feature-based systems regarding the generation and maintenance of links between sets of feature instances.

Chapter 8 Summary and Conclusions

This section concisely characterizes the state of the art and draws conclusions on the further approach of this research. It ends by recalling some of the insights from the above discussions.

8.1 Summary

Earlier scientific approaches focused on offering building blocks and automation in specialized fields (see feature-based approaches): they did not cover semantics and they did not focus on the sharing of information between applications. Current scientific approaches try to put a company's knowledge management on a more solid base by applying ontological methods for covering semantics (mainly not covering context information) in a way that is increasingly processable for machines. However, there is a trend in research toward unifying terminologies instead of allowing the chaos of heterogeneity. Such behavior is also called *ontological commitment*. The tendency to unify terminologies is even more pronounced with STEP-based solutions, which are quite common in research and which apply intermediate models. However, unification or restriction of terminology in general has been argued to be sub-optimal, before the background of this research's target to cope with heterogeneity of the IT landscape's reality.

✍ There are closed worlds of commercial software chains for supporting engineering, a world of standards trying to follow behind and worlds of sophisticated information management. Although there are certainly links between these areas of interest, there still seems to be a lack of intensive communication and cooperation. The few attempts towards an integration of information management and product engineering into an open IT concept have tried to prescribe the informational entities to be used by the participating software, thus neglecting the existence of commercial legacy software and huge amounts of product data. However, the worlds must blend into one for sophisticated future IT support in product engineering. Furthermore, assuming that all the existing applications will be replaced by new components does not seem very realistic when aiming at large companies such as automotive manufacturers.

The focus of expressive power offered by current approaches' formalisms does not match that targeted in this research. As a consequence, existing approaches offer high expressive power, which is partly not needed here and is at the same time harmful for efficiency of computation (e.g., due to a lack of encapsulation), while there is a lack of expressiveness in other areas considered crucial in this research (such as relations and contexts).

8.2 Conclusions

The above section *Summary of Conclusions from Practical Experience and Surveys* identified severe shortcomings and significant potential for improvements of the status quo in productive automotive product development. The result culminated in a catalog of requirements for new IT solutions to be deployed in this field. There is currently no research going into the same direction as this work does. The "same direction" means pursuing the same constellation of goals as stated for this work or a subset of them, while respecting the same prerequisites[*].

☞ The survey on the state of the art also failed to yield an adequate individual solution or set of solutions to the challenges formulated in the section *Problem Description*

Footnotes

[*] Formulated within item no. 4 in the requirements catalog

☞ *This chapter* states and motivates the goals of this research work and condenses them into the description of the research question.

☞ Goals of This Research.

Hence, there is no single appropriate representation formalism.

It is further concluded from the preceding summary that an informational integration able to fulfill the stated requirements should consider aspects from all of the worlds described, in that it uses a knowledge representation formalism that is flexible enough to cover all the information from existing engineering software and to describe its meaning in a machine-readable and -interpretable way. Also, any such integration should be completed by a means for online integration that offers an interface allowing running applications to share information.

Nevertheless, some interesting stimulations, as mentioned in the preceding summary, will be picked up and developed further.

8.3 Brief Recall of Some Details

Some of the major insights yielded by the above discussion are set out below:

~ Information is **context-dependent** by nature. There is no information that is universally valid.

~ **Ontologies**, in principle, are an adequate way to represent semantics in a structured fashion.

~ Although interesting and powerful information **representation formalisms** exist, there is no formalism that meets the requirements and prerequisites this research is based on. E.g., expressiveness regarding **relations** is insufficient in all formalisms. A new balance in expressivity is required.

~ Representation formalisms on an **XML basis** provide useful pre-structuring and allow the use of a range of standard tools. Furthermore, the name space concept opens the possibility of offering standardized simple data types (see *[W3C XMLschema, 2001]*). SOAP / Web Services use XML, as well.

~ There is no approach combining sophisticated information representation with a sufficient **online interface** for applications.

~ Feature linking approaches offer a restricted, specialized **automation**. As the discussion pointed out: a virtually global class model might be of use. Mapping knowledge is to be kept inside a kind of conceptual relations. Direct relations between informational entities are to be maintained. Intermediate models are to be avoided.

~ **Usability** is to be well-founded by new solutions. The approaches discussed give no direct hints on the solutions applied (see also chapter on the future work).

~ The **prerequisites for practical introduction and applicability** stated in the goals of this research are commonly not met sufficiently.

Part IV – CONCEPTS AND SOLUTIONS

In this part of the thesis a new approach is developed to reach the research goals stated in Part II – Chapter 4, and considering the insights gained from the discussion of the state of the art. This development is also based on the thoughts leading to the catalog of requirements for IT solutions for engineering formulated in the section Part II – 3.2.5, which already point in the direction to be followed now. To improve comprehensiveness, this will be done in two steps: Chapter 9 will show the major thoughts, before Chapter 10 goes into further detail.

☞ The reader is assumed to have read the brief overview before continuing with the detailed discussions starting in section *9.3*.

Chapter 9 ULEO – A New Approach

This chapter develops the major ideas of a new approach called ULEO. To allow the reader to gain a quick impression of the major thoughts, section 9.1 concisely portrays the ULEO approach, with section 9.2.2 pointing out key innovations and section 9.2.3 condensing ULEO's technical solutions. A detailed motivation and development of ULEO can be found in sections 9.3 and following. At this chapter's beginning, the research hypotheses will be stated.

The new approach is termed ***Universal Linking of Engineering Objects (ULEO)***. The motivation for selection of this specific name will be given during the detailed development of the concept.

9.1 Research Hypotheses

The general hypothesis concisely answers the research question. It will be detailed in the second half of this section.

Basically, the practical prerequisites to be fulfilled by a new solution such as scalability, flexibility, efficiency guide the way to the proposed hypothesis. They imply to manage a heterogeneous and distributed system, there no single instance database and no single source of background knowledge. It seems not realistic to be able to integrate all this distributed and heterogeneous information and systems in practice at the same time and right from the beginning. Therefore, a new solution must not demand unification of terminologies to a single one and it must not rely on a complete integration of instance information. However, it seems well realistic to document all these kinds of information in a (virtually) single information model as multiple semantic kernels. In the next step, these semantic kernels can be integrated on demand (cost/benefit) using powerful relations correlating the individual elements of the kernels – while at the same time new semantic kernels may be added to the model. From this idea of integrating semantic kernels through sophisticated relations arises also the basis for a powerful automation and provision of building blocks for engineers.

General hypothesis. A practically usable GIS that is supporting automation can be realized for automotive product development by realizing all of the following measures:

(1) **Legacy applications keep their information** processing **and are opened up for online information sharing** by code extension (e.g., via APIs) and their instance information is integrated by sophisticated relations.
(2) **Providing an online-accessible and -processable documentation** of all kinds of information in the GIS by means of general information and meta-information. There is **no unification of terminology, inconsistency** is allowed and handled; **semantic kernels** arise.
(3) Introducing **sophisticated relations** in the GIS and applying **relation-based navigation for integrating** all kinds of **informational isles. Furthermore,** these **relations** support **automation** in terms of storing **strategy information** and constellations. They represent **building blocks** as constellations. Relations of specific types control the flow of automation.

The above general hypothesis can be further detailed stating the following **specific hypotheses**. For correlating specific elements of the general hypotheses, the above numbering will be applied. All specific hypotheses must be applied together (logical AND) relation.

(1) Legacy applications can be opened up by…
   a) applying a service-based approach for online access to all types of information.
   b) externally managing relation instances correlating instance information.
(2) An online documentation of information types and control of automation can be achieved by applying all of the following measures:

a) The representation formalism supports strict encapsulation.
b) Contexts are used to handle inconsistent and application-specific information. The representation formalism supports the representation of context information for each informational entity.
c) The representation formalism and the communication services support a GIS-wide addressing concept.
d) Background knowledge can be made accessible by adding it into or correlating it to the online documentation of the application information.

(3) Sophisticated relation handling in the above sense can be achieved by applying all of the following measures:
a) Relations are entities of their own (relation partners don’t know about their involvement into relations).
b) Relations are typed and their meaning is documented (at least using natural language text) within a meta model.
c) A taxonomy of relationtypes (MTRT) is useful for building generic applications.
d) Relation types contain information on the relation partners (OO interfaces).
e) Applications know the relevant foreign or global relation types (in addition to their own types).
f) Automation can be controlled by applying generative relations (GEORs) and events
g) Strategies are represented as procedural scripts within relations.
h) Complex building blocks are represented as relations (constellations).
i) Constellations can support automation.

9.2 Brief Overview

This section gives a quick introduction to the new ULEO approach by concisely describing the basic ideas and philosophy.

9.2.1 Basic Ideas and Philosophy of the ULEO Approach

This section gives a concise overview of the major ideas underlying the ULEO approach.

A global information space (GIS) embeds applications[*] such as CAx systems by providing abstract, specific, and meta-level information to them. Each application can provide and utilize any kind of information of use. This is illustrated in Figure 15. Ideally, information is exchanged by means of an inter-process communication interface; however, file-based exchange is also foreseen in the concept.

✍ In the appendix *Coherent Glossary of Important Terms*, **abstract information** is defined as comprising concepts and relations between concepts. Adopting the terminology employed in the OO paradigm, abstract concepts are also called *classes of engineering objects (EO classes)*, while instantiated concepts are also called *engineering object instances (EOIs*). Analogously, relations are termed *engineering object relations (EORs)*. The different forms of incarnations will be explained below. In any case, relations can also relate other relations. **Specific information** comprises EOIs and instantiated relations, the *EORIs*. **Meta-information** documents the properties and meaning of relation types occurring within the GIS. Other meta-information describes context information and semantics. **Relations** are independent entities by themselves; concepts do not know about their own evolvement into relations. An application within the GIS is

Footnotes

[*] To be developed or existing

also termed a *process step application*, or *ProSAp*. Atomary pieces of information (such as concepts and relations) are also called *informational entities (IEs)*.

(I) Abstract information within the GIS. In order to document the kind of specific information a given ProSAp provides to or expects from the GIS, the respective abstract information is stored within a virtual*, ***integrated information model (IIM)***, and meta-information is stored within a ***meta-taxonomy of relation types (MTRT***, see Figure 17). Both models are accessible to all applications online via the GIS.

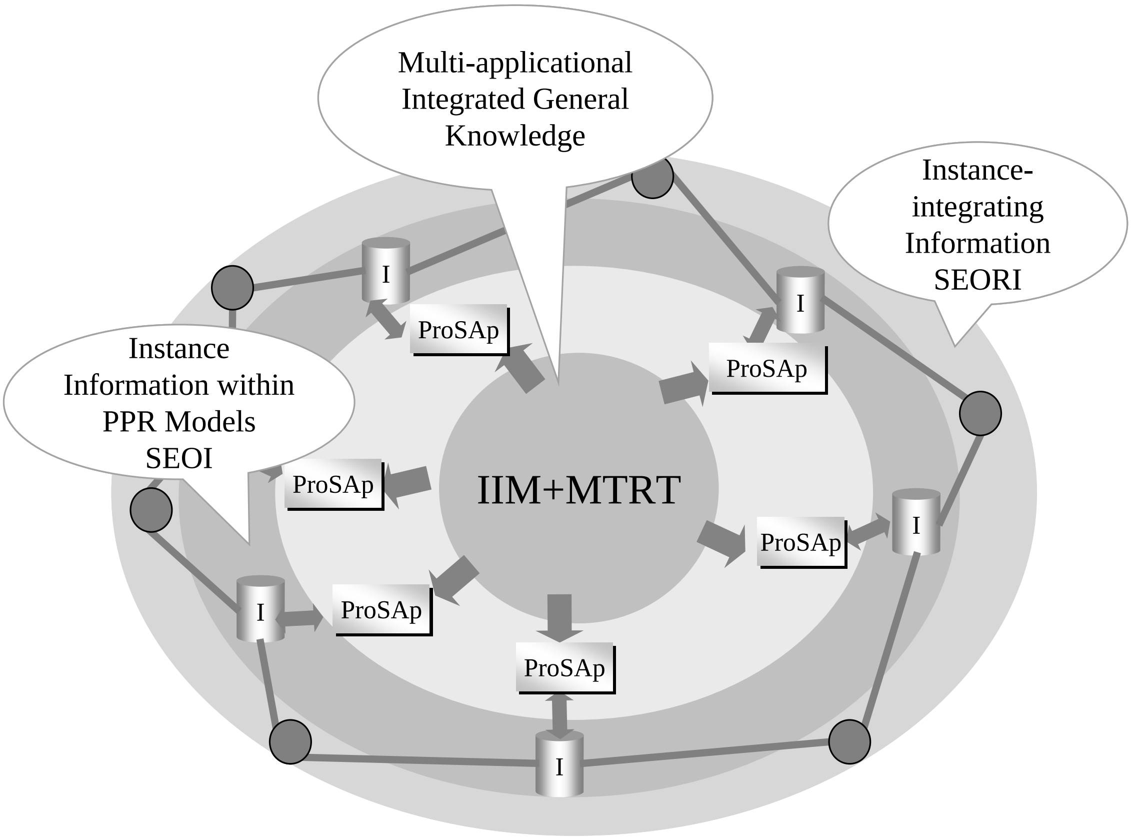

Multi-applicational Integrated General Knowledge
Instance-integrating Information SEORI
Instance Information within PPR Models SEOI
IIM+MTRT
ProSAp
ProSAp
ProSAp
ProSAp
ProSAp
ProSAp
ProSAp
I
I
I
I
I

Figure 15: Informational Integration on Various Abstraction Levels

The **IIM** hosts multiple partial taxonomies of abstract concepts, partially correlated internally and externally by relations (see Figure 17). Physically, there is a special information model called ***Unified Model of Engineering Objects (UMEO***, see Figure 16 and Figure 17) that serves as the central entrance point for retrieving abstract information within the IIM and from where references to other abstract information outside UMEO exist and can be resolved transparently. The global IIM (and thus UMEO, as well) can serve as a knowledge base in the traditional sense, in that further background information is entered in it and related to the existing concepts and relations. The IIM (in combination with the MTRT) can also serve as an ontology, which will be explained below. Nevertheless, the IIM is termed an information model and not an ontology, since its basic designated use is to document the kind of available information – but not necessarily its meaning. Semantics of IEs can be described (1) explicitly using meta-information and (2) implicitly through the interrelations between the

Footnotes

* "Virtual" means here that the model may consist of multiple physical models

informational entities. In principle, the semantics become increasingly clearer, while the IIM is filled more and more. Depending on an assessor's subjective inclinations, the model may be called an ontology from a certain point on.

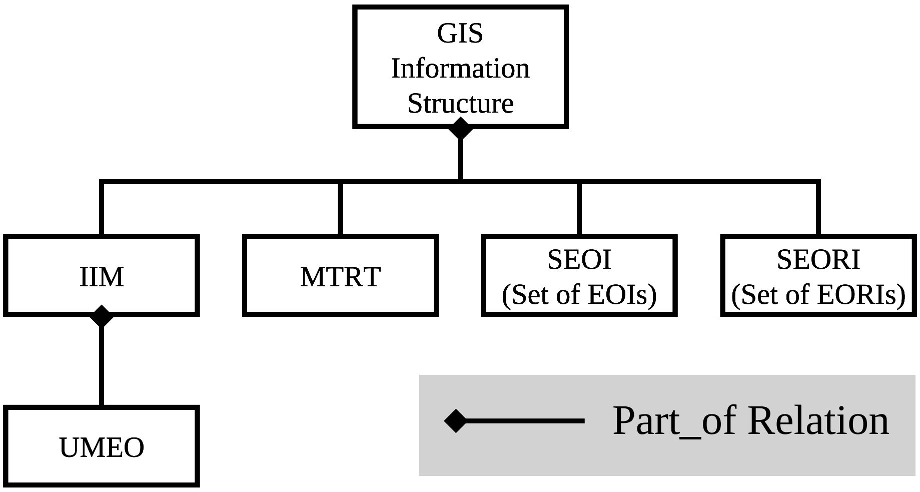


Figure 16: Components of the ULEO GIS Information Structure

Independent from these explanations, the semantics of **relation types** should generally be documented as completely and clearly as possible on a meta-level, as will be motivated below. Such documented relation types (in ULEO, also called ***EOR types***) are stored within the dedicated *meta-taxonomy of relation types.*

✍ To recall these concepts, the space of abstract information is divided into MTRT and IIM. It will therefore also be denoted as ***IIM+MTRT*** in this work. While the IIM is logically a single model, it may physically be spread across multiple models. ***UMEO, the Unified Model of Engineering Objects***, is a special model within the IIM. The complete contents of the IIM and MTRT are accessible to ProSAps online.

✍ Most of the IIM characteristics developed in the following also apply to UMEO, which is a part of the IIM. Thus, the notation ***IIM/UMEO*** will often be employed in such cases. It can be read as IIM "and/or" UMEO.

Some details on the contents of the IIM+MTRT, including UMEO. As the kind of information to be shared between applications may change frequently, any EO types and EOR types are allowed, corresponding to the kind of information individual ProSAps' may offer. The terminology is neither restricted nor globally unified: ULEO does without a general ontological commitment. In other words, no specific set of concepts is established for communication in the GIS, as is the case for intermediate model approaches such as STEP. Nor need all the concepts and relations in the GIS serve as the definitive terminology of all applications.

EO classes and EOR types may implement OO interfaces in the object-oriented sense. Thus, EO classes may inherit attributes and methods from both EO classes and EO interfaces. Accordingly, EOR types may inherit attributes and methods from EOR types and EOR interfaces. This technique is especially relevant for navigation along relations (see the section *Relation Modeling and MTRT* below).

✍ The term "OO interface" subsumes EO interfaces and EOR interfaces.

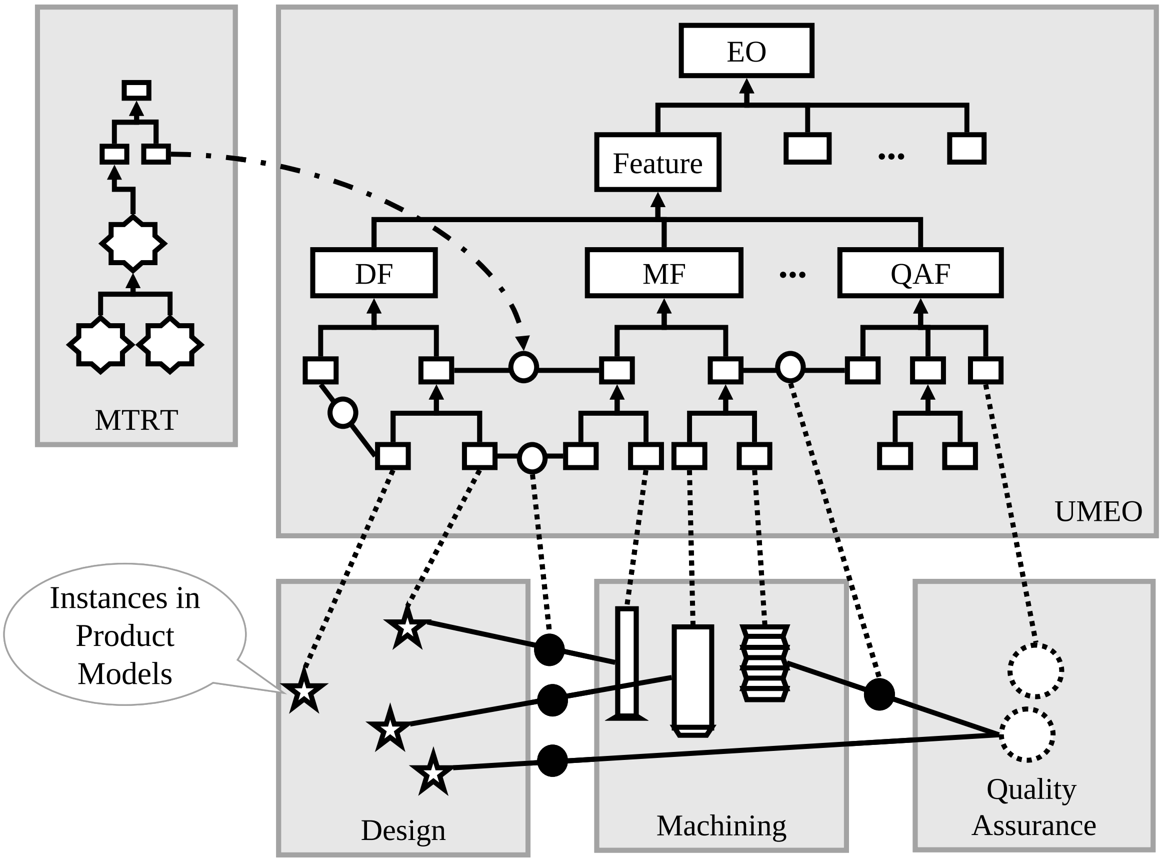


Figure 17: Partial Taxonomies in IIM/UMEO – Abstracted Illustration[*]

Utilization of the IIM and MTRT by the applications. For this, the principles of *scalability* and *use with varying degree of filling* are applied: applications knowing the meaning of model contents well enough for their tasks may utilize the model even if it does not contain detailed information on semantics. For instance, the model may contain new user-defined feature types[†] that are only of interest to a specific application. In contrast, other applications processing more heterogeneous information consume more complex information from the IIM before being able to reason the semantics and subsequently the way of processing some piece of information obtained from further applications in the GIS.

How can semantics of IEs be documented within the IIM+MTRT in spite of the missing general unification of terminology and missing general ontological commitment, implicating a reduced amount of predefined semantics? Here, the new principle of a ***semantic exploration by relation-based navigation in a meta-model***, starting off from ***semantic kernels*** is applied. To be more concrete, measures are taken to support the determination of semantics at runtime. This relies on two components: (1) IIM+MTRT must contain at least one semantic kernel and (2) the GIS and its representation format must support sophisticated representation of EORs (e.g., a taxonomy of EORs exists, and EORs are independent entities) and a detailed set of meta-information (i.e., explicit semantics, contexts). Under such conditions, ProSAps can access relevant information of thus far unknown types by following relations of known types or by possessing a documented and runtime-processable semantic description. Therefore, the existence of an MTRT covering the documentation of

Footnotes

[*] Acronyms: DF = Design Feature; MF = Machining Feature; QAF = Quality Assurance Feature
[†] Which are also kinds of EO classes

relation type semantics by means of natural language but also partially machine-processable is crucial. Each semantic kernel affects and is valid for a certain group of applications and eventually for all applications within a process step. The corresponding context meta-information reflects this. A semantic kernel arises from a restricted ontological commitment, i.e., an agreement between the applications' producers about employing a certain set of concepts and relations as their common terminology when communicating with each other through the GIS.

☞ This set of concepts and relations is only a potentially very small portion within the overall IIM+MTRT. Such ontological commitment is therefore denoted as "restricted" instead of "general". This is a major difference to the common practice of current ontology-based approaches, including those using ontology mapping. IIM+MTRT do not map ontologies by ontology but directly correlate such sub-taxonomies of concepts within the (virtually) same model.

As IIM+MTRT are open by definition, several semantic kernels can co-exist. It is desirable from the integration perspective – but neither mandatory nor enforced – that relations are created for correlating the semantic kernels' entities with each other.

✍ When agreeing on a semantic kernel, it is very helpful not merely to agree on the relation types used internally in a context's sub-taxonomy but also to define relation types leading to sub-taxonomies of other contexts, including trying to foresee relations that will be potentially relevant in the future. It is even conceivable and promising to do this GIS-wide for a certain set of basic relation types.

(II) Specific information (instances of concepts or relations). Existing software applications take on an important role within the new GIS (see the section deriving requirements for IT solutions above). Specific information within the GIS remains managed by these ProSAps as they did without the GIS. However, applications have the new – and recommended – option to document the preceding ProSAps' IEs that served as informational input for each of their own IEs constituting their individual persistent deliverables (e.g., feature instances within CAD models stored in files or databases).

✩ Example. Which specific design feature *hole* instance *A* served as input to create machining feature *cylinder* instance *B*? This may now be documented by having a relation instance created using a GIS service. This EORI is of a dedicated type such as *is_machined_as* and is stored in a dedicated logical section of the GIS that is physically realized using a database.

(III) Automation is supported within the GIS (1) by a powerful representation of relations able, for example, to host strategies and to represent complex dependencies, (2) by special types of relations able to dynamically change the degree of automation, and (3) by processing control flow in the inter-process communication interface (IPCI).

(IV) Supplier integration is supported by ULEO, if it (logically) happens within the GIS – which is rather an organizational question than a technical problem. From the suppliers' perspective, the automotive OEMs should agree upon a common GIS technology such as the one suggested in this thesis. In this context, it is also of relevance to integrate data security within the GIS IPCI.

(V) A set of **technical solutions** provides the foundation for realizing the GIS. All GIS participants agree upon the following:

~ A dedicated GIS information structure reflected in an XML-based representation formalism possessing a formal semantics and covering addressing, multiple contexts, and powerful representation of relations.

~ A set of basic data types for building complex types (e.g., XML schema data types); both may be used as attribute and parameter types; however, the set of EO classes is not standardized.

~ Online information and control flow over a service-based inter-process communication interface to assure access to information of the desired kind at the preferred time and to support automation, with the services reflecting the GIS structure.

9.2.2 What is (not) new?

This section briefly highlights the major innovation recognizable from what has been stated in the preceding section.

As a consequence of the lack of appropriate existing solutions, the following key innovations have been developed:

The idea of a **GIS** is new in that it emphasizes the importance of integrating all applications, no matter whether they already exist or are to be newly developed in the future. This includes providing all kinds of relevant information (specific and abstract, including so-called ontological information and so-called knowledge) and the need for the availability of a common infrastructure for online and offline communication, both based on a common representation formalism.

A new **representation formalism** that exactly matches the goals and prerequisites of this work will be developed: it provides appropriate, balanced expressiveness and efficiency and means for managing IEs GIS-wide – the keywords are contexts, relations, encapsulation, identification and addressing scheme.

New is also the means of **handling** the **semantics** of information: terminology will not be globally unified in the GIS, as the proposed common IIM+MTRT is not an intermediate model. Instead, standardization within the GIS will gain a new focus. This renunciation of semantic content is replaced by the new principle of **semantic exploration** by relation-based navigation in a meta-model, starting off from **semantic kernels**. Again, tackling the heterogeneity of real IT landscapes is the underlying motivation.

It is new to preserve the information management of **existing applications**: the heterogeneity in abstract and specific information is mirrored in the GIS. Nevertheless, additional means allow for full integration of application information. Also, the applications as such are assigned significant roles within the new approach.

Automation is supported by several new solutions: automation information such as procedural engineering **strategies** is stored within relationships in the IIM+MTRT, from where it can be accessed by the applications. Additionally, so-called generative relationships, **GEORs**, allow for a dynamic, situation-dependent control of the applied automation.

✍ A series of further innovations accompanying ULEO will become obvious during the elaboration and application chapters following. They are not mentioned here, as the reader requires knowledge of the basics before moving on to them.

The most important **stimulations** taken over **from existing approaches** and further developed here are the importance of **contexts** and the usage of **XML** as the foundation for a new representation format. Of course, also the basic idea of employing some kind of **ontology** to explain the meaning of facts has not been invented within this research. It has been significantly adapted and modified, however.

9.2.3 Description of Technical Measures for Realizing a GIS

To give the reader a quick and rough idea of ULEO, the more technical aspects are concisely listed below.

☞ Essential aspects of the ULEO approach are not visible from the technical perspective applied here, as this does not say anything about motivation and application of these technical means.

Please refer to the section *Basic Ideas and Philosophy of the ULEO Approach* for a more complete, but still concise picture.

A global information space (GIS) bearing all of the following characteristics:

(1) The GIS relies on a special, commonly shared information structure that is combined with and transported by a common, service-based online communication interface.
(2) Within the GIS, the existing CAx applications keep their proprietary management of instance information.
(3) Within the GIS, application-spanning relation instances are managed additionally and GIS-wide.
(4) Within the GIS, one central model, called UMEO, covering abstract information, serves as the central entry point for accessing abstract information, partially by transparent external-referencing of further abstract information in the GIS. The abstract information in UMEO together with the information accessible through external-referencing form the integrated information model (IIM).
(5) Within the GIS, one central meta-model of relation types, called MTRT, describes the characteristics and meaning of all types of relations occurring in the GIS.

At the same time, the above-mentioned **information structure** or the **online communication interface**, respectively, possess all of the following characteristics:
(6) The information structure covers abstract and instantiated concepts and relations together with relevant meta-information on the validity scope and semantics, and a common set of basic data types which is agreed upon.
(7) The online communication interface assures availability of user information of the desired kind at the desired point of time. It covers also control flow information and thus builds the basis for inter-application automation and synchronization.

9.3 Informational Integration in a GIS

This section elaborates the tackling of this research's primary goal and further motivates elements of ULEO. After reflecting upon the involvement of applications into the global information space (GIS), basic information types are surveyed, starting with a mapping of the thus far employed universal terminology of concepts and relations onto an object-oriented terminology for engineering and continuing with more detailed discussions of information handling within ULEO's GIS.

Information processing inside and outside the GIS. Embedding process step applications into a global information space is both a goal and measure at the same time. While this vision implies a solution without limits or borders between applications and a free flow of information, it also leaves space for information processing outside of the GIS. In fact, as pre-existing applications are to be integrated into the new approach, not all information processing can happen on a public stage. Rather, each application can chose its degree of involvement into the GIS according to its preferences and the benefit for the overall process chain of product engineering. Figure 18 illustrates this idea of GIS-internal and GIS-external information processing by placing applications partially into the GIS's range and partially outside it.

As a result, applications may internally manage and process information any way they prefer. Yet, from the perspective of other GIS participants, information will adhere to the documentation of concepts and relations available in the GIS's IIM+MTRT. Thus, each newly[*] developed application may decide whether to utilize the external GIS structure internally as well or to employ a different representation. Pre-existing applications' information structures may be published unchanged or

Footnotes

[*] Developed after ULEO was introduced for an OEM's product development

converted to an existing subset of IIM+MTRT contents. Such conversion ideally happens inside the applications' ULEO add-ons (see below).

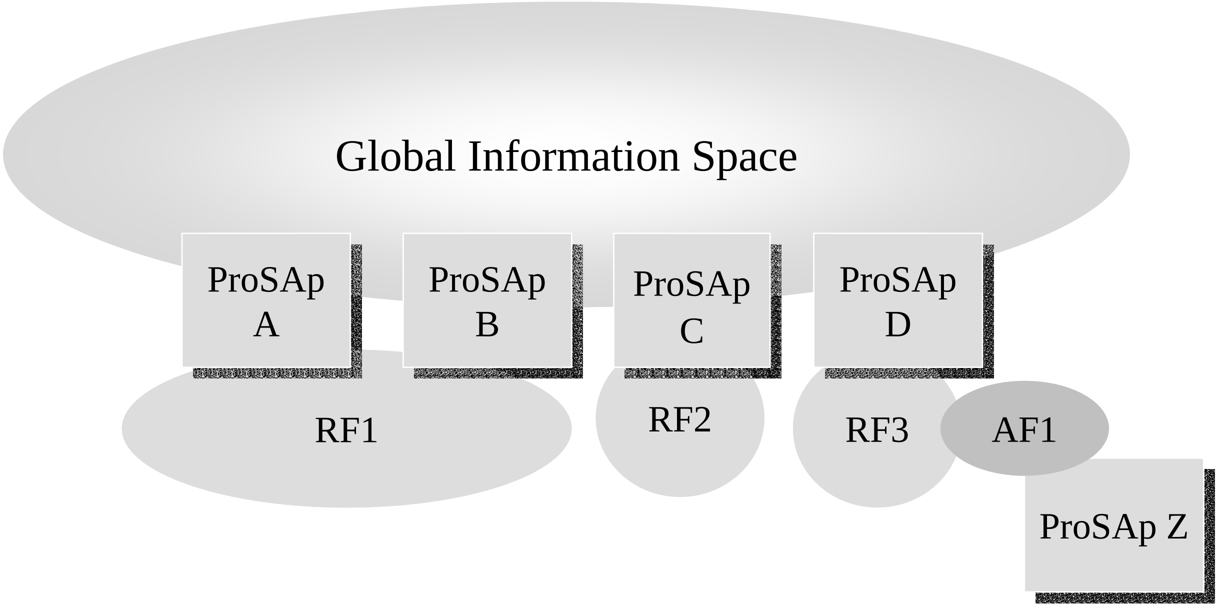


Figure 18: ProSAp's GIS-internal and -external Worlds[*]

9.3.1 Object-Orientation and Some Basic Information Types within the GIS

The OO paradigm has already been introduced. This section motivates and details its adoption within ULEO.

Adopting Object Orientation. Reacting to the requirements placed on IT solutions for engineering, first of all to achieve strict encapsulation, the ULEO approach adopts major concepts of the object-oriented paradigm as proposed and elaborated by Rumbaugh and others (see *[Gorlen et al, 1990]*, *[Rumbaugh, 1991]*, *[Booch, 1994]*, and *[Martin 1996]*). Object orientation is applied generally in state-of-the-art software engineering. By gathering attributive and procedural information around *objects*, it provides a clear structure for the information processed by software and users, which is commonly felt to be intuitive. Object classes allow for uniform and thus economical treatment of objects that are similar from the class creator's view. The subsumed methods of abstraction and inheritance allow for efficient data storage with low redundancy[†]; polymorphism supports the use of uniform interfaces between related classes. Hierarchies of object classes that are linked by inheritance relations are also called **taxonomies**.

Features are probably the most prominent kinds of objects in product engineering. How features are modeled strongly influences the resulting benefits of feature technology. It has a direct impact on the quality of the information that can be represented inside PPR models and materializes in terms of

Footnotes

[*] Acronyms: RF = (proprietary) Representation Formalism; AF = Alternative Representation Formalism

[†] Which facilitates maintainability

the collection of feature properties, relations between features, and the meta-information on both. Other quality criteria are extensibility of models and the option to integrate them into other models, absence of redundancy, availability of building blocks for new features, and user-definability of features and relations.

An extensive survey of the literature on feature linking (see above) showed the common usage of taxonomies in more recent approaches. Shah proved the object-oriented approach to be suitable for feature modeling (see *[Shah et al, 1990]*). But it is not sufficient to focus on certain types of objects inside the product development domain such as features or parts. Instead, all kinds of information that are possibly relevant inside a company must be regarded. In order to denote all the objects relevant in product engineering, the notion of **engineering objects** (**EOs**) is introduced. A similar step has also been taken, for example, within VDI guideline 2218, which replaces the notion of feature technology by that of an object-oriented product modeling (see *[VDI-2218, 1999]*).

Some details. Modeling engineering objects in the object-oriented sense means to differentiate and arrange them into EO types or classes. After that, EO classes can be instantiated into product models (→ EO instances, EOI). An EO class may consist of attributes, which describe the static properties, and sometimes also of methods, which describe the dynamic behavior of the respective EO instances. To reduce redundancy among EO classes, one or more steps of abstraction are typically performed where common elements of some EO classes are extracted and conglomerated to a new EO class called the parent class, which is related to the child EO classes through a kind of relation called an *is_a* or *specialization* or *inheritance* relation. Such relations are hierarchical in the sense that a parent class bequests all its properties to its child classes. The result of finding EO classes, performing abstractions, and arranging the classes by linking them through specialization relations is a taxonomy of EO classes. EO classes on the bottom level of the taxonomy tree can be instantiated by the user, whereas classes located higher in the hierarchy cannot in every case. The latter are also called *abstract classes*.

GIS terminology. The basic informational elements in ULEO's GIS have been introduced briefly. In principle, they are abstract and instantiated concepts and relations. Both concepts and relations are objects in the OO terminology, in that all of these informational entities may carry attributes and all (concept or relation) classes may carry methods as well. Abstract entities are called *classes*, instantiated ones *instances*; thus, there are classes and instances of concepts and relations. Using the terminology just introduced, *EO classes* are classes of concepts, and *EO instances* are instances of concepts. Anticipating the introduction of EO relations, classes of relations are *EO relation types* or, in short, *EOR types*, and instances of relations are *EO relation instances*, or, shorter, *EOR instances*, or simply *EORIs*. This EOx terminology is used preferentially within the ULEO approach.

As all of these informational entities are considered to be objects, they are strictly encapsulated when occurring with the GIS, i.e., represented using ULEO's representation formalism, called **ULEO XML**. "Strictly encapsulated" means that all defining information (inner properties consisting of the set of all attributes and methods) on a given IE is fully contained within a single dedicated representational element, which in ULEO XML is an XML element. Information on relations (outer properties) to other IEs is not included in an IE's description and is not considered part of the IE's inner properties. An IE is not informed about its outer properties. Outer properties are represented by means of relations. Please refer to the section *Relation Modeling and MTRT* below for a motivation.

☞ Concepts and relations have very different semantics.

9.3.2 Abstract Information in the GIS

This section clarifies the question of how abstract information should be managed and processed within the GIS.

Brief motivation. Although object orientation lies at the core of today's practical IT solutions, it is not really adopted in its full scale in IT for engineering: although the instance-level information is intensively processed and managed by all systems, the complementary class information (also called **abstract** or **background information**, see above) remains hidden inside software code. While it is hard to tell how the meaning of instance information may be fully understood without it, it is not commonly accessible – neither to software engineers nor to applications or end users. Consequently, considering background information is also a prerequisite for facilitation of a high-level information flow and a global information space. As will be argued in the section on *Automation* below, managing background information is also a powerful means of handling automation knowledge. Therefore, this issue meets several of the above-stated requirements.

✍ The application of OO interfaces to EO classes and EOR types is discussed in the section *Relation Modeling and MTRT*, as this is of special relevance for EOR processing.

Integrated Information Model and UMEO

From what has been argued above, it is concluded that object-oriented **information models should be available online to any process step application** participating in the future product development process chain. Such models should describe the nature of all informational entities processed by a certain application and relevant to other applications (ProSAp's view on the domain[*], see the section *Views on Informational Entities*). Thus, for a given company's process chain, **correlations** between the ProSAp's EOx can be detected and recorded by relations. **Relations** are the key to integrating information formerly handled separately inside the individual applications and to modeling automation knowledge. Accessibility of all relations to any application is the foundation for achieving *multi-directional and also domain-spanning associativity.* Using relations, applications are able to access other applications' information and may share specific instance information and automation knowledge.

These considerations suggest to maintain an **integrated information model (IIM)** where all informational entities (IEs) that are of common (i.e., inter-application) interest are described and correlated. By mirroring the ProSAps' original sub-taxonomies in the IIM, the above-stated demand for **task-oriented optimization** of informational entities will also be supported[†].

✍ **Replacing existing applications' information structures**. Reuse of existing applications' information structures and instance management is promoted. However, the GIS information structure should (partly) replace those of commercial applications that are not structured or expressive enough. A company risks losing its know-how under such conditions. A prominent example is a major CAx software vendor's *Knowledgeware*, which spreads many small pieces of knowledge all over the proprietarily arranged product models. There is no effective possibility for a company to keep track of the know-how represented in such structures, nor is there a means of maintaining it. The same holds true for knowledge hidden inside automation scripts that are stored on some intranet drives. When partly replacing existing applications' information structures, the associated algorithms and elements of user interfaces have to be reimplemented as well.

☞ This IIM+MTRT approach is contradictory to state-of-the-art approaches to standardization of product data handling such as STEP. While standard formats force applications to convert their own data structures into standard ones, the approach suggested here is to publish such proprietary structures unchanged and to subsequently integrate them using EO relations. Benefits are an unrestricted information flow (direct relations and no lack of expressiveness as there are no

Footnotes

[*] "Domain" is used here in the sense of "step within the product development process"

[†] The discussion of domain-specific feature-types can be directly applied also to engineering objects.

intermediate formats[*]) and best possible integration of existing applications into a GIS. Relationships can be defined and created as they best fit the respective process chain. The drawback on the other hand is the need for manual integration of new ProSAps. And, although a company-specific information space can be tailored to the company's needs, it is not necessarily directly transformable to that of other companies. Also suppliers have to be integrated manually. During the introduction part of this thesis, the importance of reusing legacy software has been pointed out. But also from a theoretical point of view, such **integration by individualization instead of homogenization** seems to lead to higher performance from the individual companies' perspective.

As a first improvement of the situation today, software applications can be extended on the basis of a hard-coded IIM+MTRT variant, to consider other applications' information. While this hard-coded solution can provide high-level integration, it is, of course, not very flexible. Hence, a second level of improvement suggests itself, forming a very flexible solution: storing the **IIM+MTRT explicitly** in a **machine-interpretable** form and **freely accessible** to all ProSAps, as it is common for instance information. Now, any ProSAp can retrieve, interpret, and dynamically react to the contents of the integrated information model. Starting off with their own IEs within the IIM, the ProSAp can follow familiar relations to find EOx it is interested in. Depending on the kind of relation followed, pieces of information that are relevant in a certain context can be retrieved.

☆ For example, several design feature classes could be linked by EO relations to form a pre-defined, but dynamically changeable building block for instantiation within a single user action (automation). Another example is information reflecting user experiences that is provided to help other users when utilizing certain feature types. IIM/UMEO provides the technology for *handling users' experiences* very efficiently. Among the first to investigate this kind of information were Böhle and Rose (see *[Böhle & Rose, 1992]*).

☆ Another example: An advisor software assisting experts in selecting inspection strategies may use the IIM's inheritance relations at runtime, in that it presents help information actually specified for specific classes only, for their child classes, as well. This is only possible through an information model that is explicit and accessible online.

Adopting this flexible method generally in the entire product development of a company allows for replacing applications with others without having to change those remaining significantly. The identity or exact sets of attributes of EO classes are not fixed, and new types of EOs such as certain features may be added and processed. Additionally, building blocks for flexible automation can be added or changed. In the same way that ProSAps may access the IEs within other ProSAps' sub-taxonomies, they can also access company know-how stored within the IIM. In fact, there is no qualitative difference between these kinds of background information (knowledge, know-how). Thus, company know-how is naturally integratable into the IIM, making it generally available and usable. This includes knowledge about standard procedures such as best practices, common errors, or guidelines from management.

✍ It is not suggested that knowledge of the entire world be managed, as some existing approaches do (see *[Lenat, 1998]*). Instead, a certain field is to be targeted, the product development. Although the approaches developed in this work do, in fact, work for other and larger application fields, too, it seems to be realistic to cover most of the domains' relevant background knowledge inside the IIM.

Footnotes

[*] See also the discussion of state of the art for feature-linking approaches.

☆ Further example: the IIM+MTRT also takes over the role of a data dictionary and an online documentation of the informational entities currently used inside the software applications involved in the GIS. Hence, EO classes and EOR types used inside a CAD system for detail design and relevant for being shared within the global information space are entered into the IIM+MTRT.

One-Entrance Principle and External Referencing

In principle, the IIM can be obtained by relating physically distributed information models or by relating several partial taxonomies within a single global model (centralized[*] approach), or by a combination of both.

The existence of a central entrance point (UMEO) for accessing all GIS applications' background knowledge assures an optimal overview of all available information (**one-entrance principle**[†]). Furthermore, correlating as much connatural information as possible within the same physical model raises its availability and avoids redundancy.

✍ In addition, storing as much information as possible within one place and using neutral formalisms also allows addition, removal, and replacement of the applications responsible for processing this information. This *separation of information and algorithms* makes company know-how better accessible and more universally usable (see also knowledge-based approaches), i.e., usable in more different ways and by more different applications.

✍ The decision as to whether to integrate a given information base into UMEO also depends on the complexity of its contents in terms of the kind, cross-linking, and amount of its informational entities.

On the other hand, for historical and organizational reasons, it might not be easy to maintain all the abstract information in the GIS within a single model. This seems especially unrealistic, if much information already exists in a company and if this is spread over many knowledge bases.

For the reasons cited above, it is suggested to adopt a hybrid approach of defining a specific information model within the IIM to be the GIS applications' central entrance point when accessing abstract information. At the same time the IIM is to be defined, in principle, to be virtually one model that may be physically distributed onto several information models, with these distributed models transparently correlated to the central entrance model by means of **external referencing** through EORs.

✍ Choosing this solution allows the benefits of a centralized[‡] approach to be exploited, while at the same time avoiding the unrealistic claim of being able to cover any amount of information within a single model. Also, the existence of a central entrance model within the IIM does not proscribe all accesses to other models.

☞ Such incarnation of a central entrance information model in the GIS will be called **Unified Model of Engineering Objects** (**UMEO**). Note that the term "unified" within UMEO does not stand for a unification of terminology – which has already been rejected above – but shall intimate (1) the fact that UMEO comprises and correlates multiple non-unified terminologies within a single model

Footnotes

[*] "Central" means that there is only one taxonomy (comprising sub-taxonomies) for all the ProSAps, and the taxonomy is stored in only one globally accessible location

[†] Today also commonly applied for Internet portals

[‡] This has also been stated to be favorable during the conclusions to feature linking approaches.

and (2) UMEO’s special role resulting from the one-entrance principle. Nevertheless, UMEO might have been also called "IIM entrance (sub)model" (IIMEM).

As has been intimated, it is beneficial to fill UMEO well with information. Nevertheless, it is highly advisable to correlate UMEO contents logically to the other contents within the IIM, thus, for example, opening access to third-party applications’ abstract information. This correlation can be transparently achieved by an external referencing mechanism based on EORs.

✍ The adjective “transparent” is taken from computer science terminology and means “not noticeable by the using ProSAps”, i.e., the external referencing happens in the background and is hidden to applications accessing the information. ProSAps following such external references will not notice – if they do not explicitly ask about it – that information is outside UMEO and that it has been accessed via external references.

☞ In order to maintain the one-entrance principle, there are no third-party applications’ informational entities that are not either directly contained in UMEO or correlated to it via the external references[*].

Transparent external referencing is facilitated by having the GIS services (i.e., IPCI services) perform the external calls automatically on retrieval of EOx attribute values or of EOR partners. Such information retrieval services call reference resolution services, passing on to them the external references. External references may be encountered within the EOx attribute values mentioned or within EOR partner specifications and are formulated as IPCI service calls. On retrieval of the whole EOx class contents, such calls should not be performed automatically but instead by explicit request. Thus, it remains possible to access and manipulate these service call entries. The same rules apply for external referencing from within EOx instances.

The Role of IIM+MTRT as Ontology and Knowledge Base

The more sophisticated information is stored within the IIM+MTRT, the more IIM+MTRT can take over the role of an **ontology**: in addition to the individual entities’ explicit semantic descriptions, it represents their semantics by their embedding set of other EOs and EORs, which is the typical purpose of ontologies. Thus, one part of the semantics is explicitly documented inside the entities themselves while another part of it is represented indirectly (implicitly) by the network of relationships of the informational entity.

As the IIM is interpreted by ProSAps to control their own behavior, it can also take over the role of a ***knowledge base*** (see also the demand for flexibility of software).

☞ Nevertheless, it has to be explicitly pointed out that the IIM+MTRT plays an important role even if not filled with sophisticated knowledge. Thus, IIM+MTRT’s usage by ProSAps is *scalable* in a wide range.

Footnotes

[*] In addition, all types of relationships used there must be stored inside MTRT. See below.

Relation Modeling and MTRT

In order to achieve a powerful integration of application information processing, a ***sophisticated relation concept****, including the representation and management of relations has been argued to be crucial (see section Part II – 3.2.1 above). This idea will be further motivated in the following.*

✍ It might be of interest that the vital relevance of relationships can be assumed to be very generally valid for all areas of life. Robert Nozick even suggested understanding the meaning of life to be no (meta-physical) object but a relation, i.e., a combination or constellation correlating objects and other relations (see *[Nozick, 1990]*).

Returning to the details of the more concrete field of product development, relations must not be limited to geometry-oriented dependencies, but must also be capable of representing any conceivable correlation between informational entities (see also Figure 19). Consequently, they must not be restricted in the maximum number of relation participants (**arity**)[*]. Hilko Koopman performed some investigations on the role of relationships for product engineering during his diploma work (see *[Koopman, 2004]*).

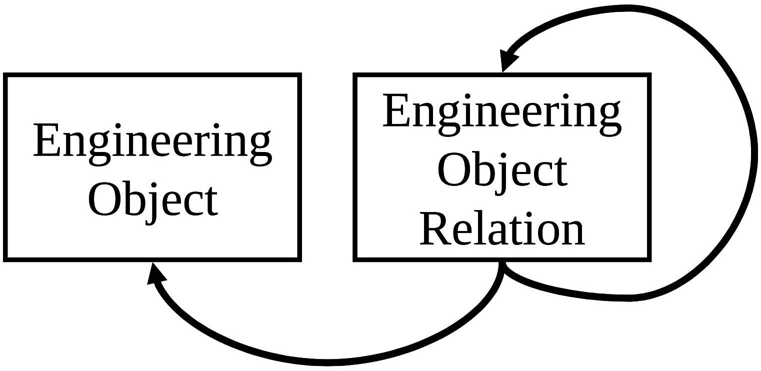


Figure 19: Meta-Schema: EO Classes and EO Relations

To further emphasize this **relevance of relations**, it should be considered that not only do relations embed concepts into a network of other concepts in order to make their semantics accessible, but they are also a significant part of the information by themselves. To be more precise, they correlate concepts to form **constellations**, and they represent **transitions** between domains in product development – which illustrates that relations are not limited to representing static dependencies but also extend to methodical and thus procedural ones. Both occurrences possess logical correspondences in product development: the latter represent strategies and procedures yielding an output by consuming some input; the former are combinations of multiple concepts and/or other relations, representing abstract (but relevant) constellations. The conjoined informational content of such constellations forms a special context for the individual relation partners involved, and by itself constitutes an independent entity as analogously relevant as concepts. The *Quality Criterion* relation, to be demonstrated during the *demonstration* part of this thesis (see section *Part V – 13.3.1*), will show this clearly, in that it became a central focus element of quality assurance processes for I++

Footnotes

[*] This will be shown in the I++ information model, where many relation types have four or more partners.

members. Overall quality assurance is based on and correlates its results to quality criterion relations – therefore, relations must be represented as objects and (at least) equally detailed as concepts. Here, constellations represent handles for a set of information wrapped for a special purpose. Of course, each component within such bundle can occur in multiple other bundles as well.

Referring to the discussion of **feature linking approaches**, the set of expressible relations is no longer restricted to inter-feature relations. Instead, relations between feature components (intra-feature links) are also representable, as well as relations between features and other entities such as parts, assemblies, and resources. For instance, relations can be naturally and efficiently applied to represent information on product variants. This increases the value of the product models significantly, as these relations facilitate assembly modeling (e.g., assembly features) and assignment of resources and processes, which are key milestones in building a bridge to the logistic process chain. Feature linking is generalized into a ***Universal Linking of Engineering Objects (ULEO)***.

✍ The **term** *Universal Linking of Engineering Objects*, or short *ULEO*, is used to denote the overall approach developed during this research. The method of linking logically related engineering object classes and instances is regarded as the key for both informational integration and automation. Although the approach defines more than merely linking objects, this term emphasizes the crucial importance of a sophisticated relation concept within a GIS and points to product engineering as this approach's ultimate destination field. The term "universal" claims that the application of ULEO is not limited to certain domains in product development.

Further benefits. The resulting greater expressiveness for representing relations allows for expressing dependencies that are not representable otherwise. More sophisticated informational entities are representable and usable, yielding positive effects on the entire process chain in terms of being closer to the real tasks that engineers have to fulfill. To give another example, this also enables elegant handling of **users' experiences** regarding certain EO instances and EO classes and is a base functionality for offering user **assistance** functionality. Also resulting from the higher expressive power, modeling feels more natural in many cases. Although modeling is non-deterministic, the possibility of intuitive modeling is a hard fact for powerful information structures since information may be stored, retrieved, and presented the way the domain experts expect it, i.e., based on the concepts their specialist knowledge comprises. Thus, the expert is likely to be more willing to enter own knowledge and is more likely to understand knowledge from other domains. Increased usability of IT has been stated above to be one of the requirements for new IT solutions.

Realizing a Sophisticated Relation Concept

To realize such a sophisticated relation concept, basic informational entities in a domain have to be treated less uniformly than concept-only approaches do, as every domain consists of entities playing the role of building blocks[*] (concepts), while other entities function as links in-between (relations). For integrating domains, relations are even more relevant than concepts as has been shown above. Thus, it has been suggested that relationships between entities be modeled separately from these related entities by means of ***engineering object relations (EORs)***[†]. EORs are considered and treated as equally important entities. This means that EORs have to carry all the information comprising a given relationship, i.e., they have to know about all the relation partners (IEs participating in the relation) and they have to carry attributes and methods describing their properties on a sophisticated level.

Footnotes

[*] For example, for building product models

[†] Similar to what is termed *association class* in UML, but more elaborate and expressive (see below)

Another major design decision within ULEO concerns **engineering objects** (classes and instances) and is a logical consequence of the idea of strict encapsulation of all relational information within EORs: EOs do not know about the relations they are involved in, although the applications handling them are, of course, able to retrieve this information. Consequently, an individual engineering object's inner properties (object-defining properties) are not influenced by the existence of any relation that this EO is a member of, and EO classes do not have to be adjusted each time new relation information comes up.

☞ Relations cannot be replaced by concepts without losing expressive power, as a relation represents all the information relevant for a certain relationship in one IE, whereas a concept does not know anything about its partners. Therefore, if no dedicated relations are available, concepts refer directly to other concepts via attributes containing other concepts' identifiers. For example, if an EO class is involved in more than one relationship, this can be represented by several EORs. In the absence of EORs, the respective EO class refers to several other EO classes instead; but which of the "partial" relations belong to which super-relation is not clearly representable. This unstructured set of pointers (IDs) can be structured by using fixed data structures as members of the EO class. However, this is an inflexible solution and not suited for dynamic information models. As a relation usually affects at least two EO classes, changes in the relations have to be performed in both classes, which is, of course, error-prone.

✍ Although there is no clear rule for **classifying an IE to be a concept or a relation**, a rule of thumb can be given based on the fact that a relation refers to all of its partners[*] and cannot exist without partners, whereas EOs do not know about other EOx at all: if it is possible to describe an informational entity without referring to other IEs, it is a concept, if not, it is a relation. In cases of doubt, the following rules may help: (1) Relations are non-physical entities, i.e., they, for example, do not have geometry. They are not direct product components but may describe relations between product components. (2) Physical objects such as product components are always concepts.

Relation Types and OO Interfaces

Classifying relations provides a clear structure inside the relations' network; their types are arranged as engineering object relation classes, also called ***EOR types***, that describe their respective characteristics. As applications can follow types of relationships and are not fixed to individual relation instances, access to information happens more efficiently. EOR types help to avoid overcharging the IIM while representing any number of manifold logical correlations between informational entities. That is, their application leads to a multi-layered network where information is stored locally in the sense that dependent entities are directly related. For example, *company know-how* is not isolated from other information but locally linked to it (see also the discussion of the current state of product development). The result is a clearly structured, targeted, and maintainable information space as called for in the catalog of requirements stated in Part II – Chapter 3. EOR types allow **filtering**, which is the key for manageability.

✍ The cross-linked EO instances can be utilized by ProSAps to provide context-specific information to their users. ProSAps can retrieve the desired task-relevant information by following EORs on the instance level. They target EORIs of certain types that are pertinent for the specific purpose and disregard others.

Mandatory properties of potential relation partners (IEs correlated by a relationship) are definable within each EOR type by using object-oriented **OO interfaces**: due to this feature, ProSAps can trust

Footnotes

[*] That is, to all the EOx involved in this relationship

in finding (at least) these sets of attributes and/or methods within EOx that they have reached by following an EOR of the respective type. As a result, applications can dynamically follow relations of the desired types and utilize the properties given by OO interfaces. OO interfaces can be arranged within an own *meta-taxonomy of interfaces* or within UMEO as special kinds of EO classes. In the former case, it is not necessary to maintain separate taxonomies for EO interfaces and EOR interfaces, however.

✍ Although the UMEO-internal solution may be regarded as a border-line case, since OO interfaces do not necessarily describe full objects but only properties of them, it reduces the complexity of the information space. Moreover, this is a natural way to allow the specification of whole EO classes as OO interfaces, thus prescribing that the respective relation partner has to be a certain EO class or its children.

It is hence suggested that OO interfaces be managed inside UMEO. In any case, it is necessary to document the OO interface semantics inside each OO interface description.

By using a taxonomical model for storing EOR types and additionally publishing this model equivalently to the IIM, **inheritance between EOR types** can be used by ProSAps at runtime. Thus, ProSAps can utilize not only relation types known to them at their coding but also all the children derived from them and added to the taxonomy later. This is a straightforward way of deducing new, explicit knowledge from the implicit. Generally applicable information is attached to parent classes, while respective exceptions are attached to their children. As this model of relation types contains meta-information on relationships, it will be called ***Meta-Taxonomy of Relation Types (MTRT)***.

MTRT and IIM. Resulting from the given approaches, relation types within the MTRT are an abstraction of relations inside the IIM. Each MTRT relation type can occur any number of times within the IIM. The occurrences of relations within the IIM will be called ***engineering object relation materializations (EORMs***, see Figure 20). EORMs may link classes of EOs to other classes of EOs, or link classes of EOs to EORMs, or correlate any other combinations of EO classes and EORMs. However, as the IIM does not contain any instances of EO classes or EOR types, EORMs never link EOx instances.

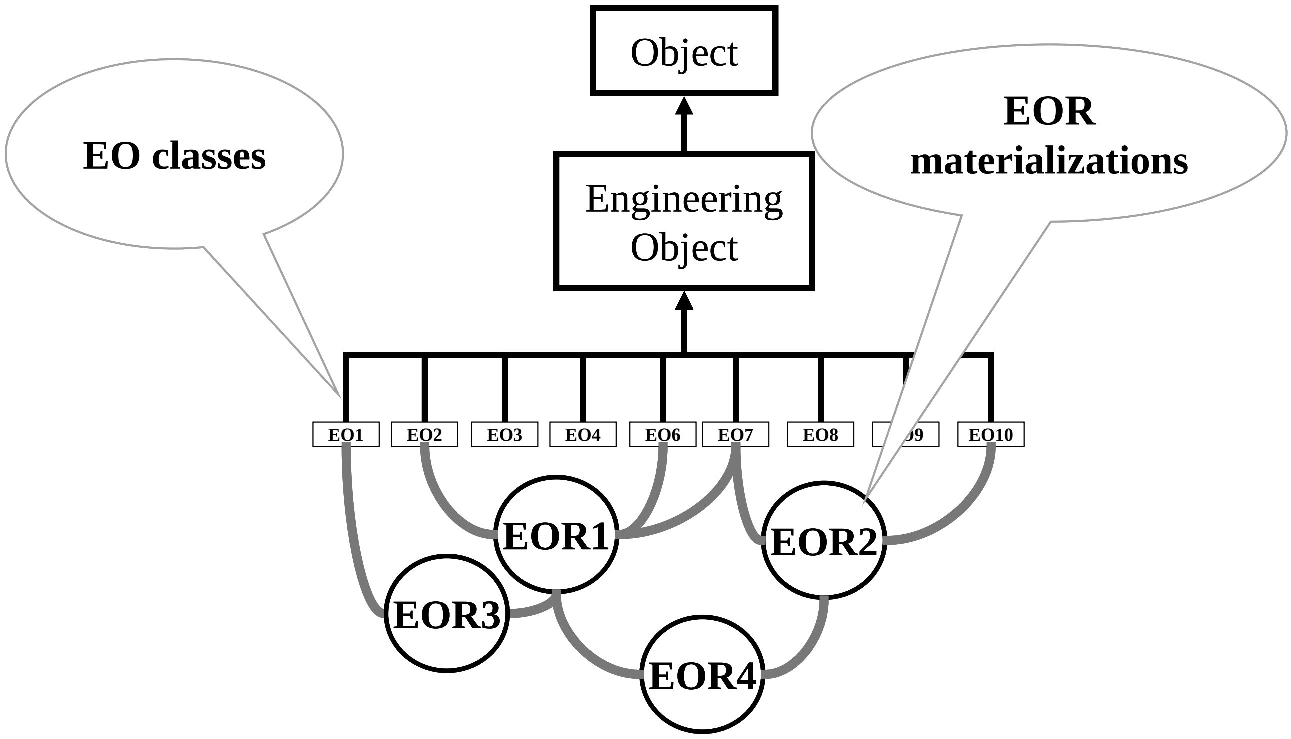


Figure 20: EO Classes and EOR Materializations within IIM/UMEO

The effects that **inheritance relations** between EO classes yield on the EORMs that these EO classes are involved in can be specified within the MTRT as part of the semantics description for each EOR type (a special kind of meta-information).

Various ways of handling are conceivable: while EORMs related to child EO classes could replace the parent EOs' EORMs if they are of the same or a subsuming EOR type, it may also make sense in some cases to keep using both. Generally, information on a higher level of abstraction (i.e., higher in the taxonomy's hierarchy) can be thought of as default information, while information lower in the hierarchy describes special cases. Thus, frequently made user experiences (attached to lower EO classes) may be lifted up to express their generalization.

To sum up, engineering object relations are a kind of specialized entities that carry special attributes and methods and are stored and processed especially to promote clear-cut and sophisticated handling of relationships.

Relation Instances Integrate PPR Models

The significance of engineering object relations for integrating ProSAps has been shown. As a consequence of the fact that current CAx software does not know sophisticated relations (instead they are uni-directional and geometry-focused), EOR management has to be put on the process chains applications from outside. Yet, this is the only possibility to inter-relate EO instances spread over multiple PPR models: relations correlating EO instances and spanning applications do not belong to a single product, process, or resource model but – as a matter of fact – lie between these models.

Based on this situation, the following approach is suggested: ***engineering object relation instances (EORIs)*** are stored outside all PPR models and inside a **global EOR instance database** handled by a central inter-application EORI management as part of the global GIS services. Resulting from the decision to separate EOs from EORs, this can be easily achieved as EOs do not know about their external relationships* and thus do not have to be modified. The EORI management has to be central in order to ensure the retrievability of all EORIs (one-entrance principle, UMEO), which is crucial to be able to provide a global information space. Additionally, a distributed EORI management would become very cumbersome as EORIs will form a large and dense network in practical use. Thus, one global network of relations correlates the de-centrally stored EO(R) instances.

✍ Proprietarily handled relations inside PPR models will not be replaced by central EORIs if they are made accessible to the GIS – this is a tribute to the requirement to preserve proprietary data structures.

The resulting **benefit** is an integration of PPR models on any level of detail (controlled by the type of EO instances) and of sophistication (controlled by the EOR types used). As PPR models reflect the views of individual process steps on the same product, process, or resource (domain views, see also the section *Views on Informational Entities*), logically connatural information can now also be correlated physically. See Figure 21 for a schematic illustration of IIM/UMEO, MTRT and EOx instances.

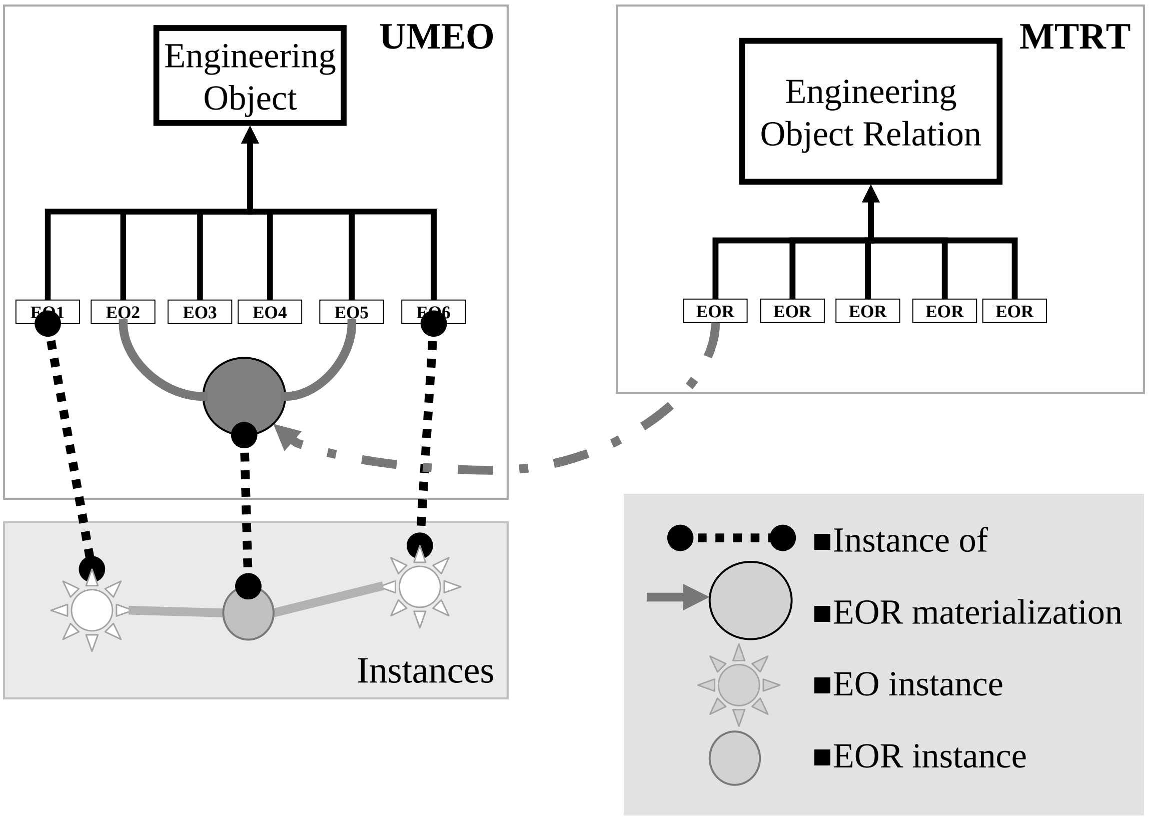


Figure 21: Basic Information Types in the GIS (schematic)

Footnotes

* Although the applications are able to retrieve them

IIM/UMEO and MTRT Services

IIM/UMEO and MTRT have to be made **persistent** by some form of physical data storage such as a database. In order to abstract from the database's physical implementation (database schema), the introduction of a software application that is accessible GIS-wide by GIS IPCI services is proposed. The two groups of services will be called ***MEO services*** and ***MTRT services***, respectively. The former also comprise the above-mentioned **external referencing** of IIM accesses.

Information Acquisition – Filling the Information Base

Filling the information base with background information (knowledge) is one of the most prominent and critical issues when practically applying knowledge-based approaches. Two principally different approaches that shall be discussed briefly here are conceivable:

The ***top-down method*** starts off from the very root EO class (e.g., called *AnyObject*) of the taxonomy and develops the EO class taxonomy "downwards" by specifying all relevant child classes and continues working this way for the children of the children until the domain is fully analyzed and represented within the information base. On each abstraction level, the top-down method tries to fully cover all children of a class, thus critically depending on a sufficient degree of holistic thinking and a global overview. This approach, however, may cause very time-consuming analyzes in case of complex domains – as product development is – thus requesting for long introducing periods of new software. In fact, it may turn out to be almost not feasible. This becomes even clearer, if one recalls that also any kind of relationship between the entities has to be covered.

The ***bottom-up method*** starts locally by analyzing small, limited sub-areas of the domains, resulting in small taxonomies. As the area of interest is manageable, results can be achieved in rather short periods of time. Domain experts' knowledge can be inquired by interviews led by knowledge engineers. However, applying this method for several domains yields several un-coordinated taxonomies, as they do not base on a common root.

For the given reasons it is suggested to combine both methods by starting off with the top-down method, however not requesting completeness of the model but covering just the most relevant topics instead, in order to be able to provide a solid foundation for the subsequent application of the bottom-up method. Thus, top-down modeling assures a well-structured information model. Of course, the whole process will be repeated in any number of loops by detailing more abstract classes by less abstract children. The exact degree to which the top-down method should be initially applied depends on the manageability and clearness of the domains and is finally a matter of weighing up costs against benefits.

Sources of knowledge are IT experts investigating existing applications' information models, some selected domain experts and a larger number of "common" engineers, entering their experiences during their everyday work. Also EBok[*] systems and other advisory systems' information bases can be integrated into the global information space. However, as already pointed out, re-entering such information might be more promising, as it can be directly related to existing ProSAps objects in the information model.

It has already been pointed out that IIM/UMEO works fine and brings benefits almost from the first beginnings, not depending on a large fill level (see demand for scalability).

✍ As already explained during the discussion of the state of the art, it is assumed that the integration of new entries into the IIM and their correlation to already existing ones should be – at least in the general case – done by a human, understanding the whole affected context; and it should not be done looking at a single application only.

Footnotes

[*] Engineering Book of Knowledge

✍ Please refer also to section *Process of Filling UMEO – Information Modeling* below for pertinent practical experiences.

9.3.3 Specific Information in the GIS

Specific information, which is called instance information *in the object-oriented paradigm, is that part of the PPR information, systems in product engineering have traditionally focused on. This section further motivates its placement within the ULEO approach.*

Today's product models are built-up of EO instances and of relations between them. It has already been argued in the preceding sections that *integration by individualization instead of homogenization* is the preferable approach to reach the stated goals as it allows full utilization of individual ProSAps functionality and best optimization of inter-ProSAp cooperation according to the company's needs. Hence, analogous to the publication of existing applications' unchanged information models, the applications' instance information structures must be preserved: ProSAps keep use of their proprietary information management and manipulate data in the same formats they have done so far. In ULEO terminology, this means that ***instances of engineering object classes (EO instances, EOI)*** are stored and managed by ProSAps themselves.

✍ This decentral solution produces a very positive side-effect: it meets the requirement for future IT for engineering calling for reuse of existing software and data in order to achieve scalability.

☞ As the class type of EO instances within those PPR models is known, there is **no** explicit **link** needed between EO **classes** and EO **instances**; EO class identifiers will be used, instead.

For being able to exploit relevant IIM/UMEO contents, and to dynamically react to their changes, an existing ProSAp's functionality has to be extended appropriately. Most of today's CAx systems allow for achieving this by customization via APIs or scripting.

9.3.4 Brief Interim Summary

In order to ease following the development of the ULEO approach, some technical issues are very shortly summarized here.

Some of the current technical issues are reviewed below:

~ MTRT stores EOR types.
~ UMEO is part of the IIM.
~ IIM/UMEO stores EO classes and EOR materializations.
~ EOR types from MTRT materialize into IIM/UMEO as EORMs. Materialization is a semi-instantiation where some – but not all – EORM attributes are assigned values.
~ EO classes are instantiated by ProSAps into their proprietary PPR models as EO instances.
~ EORMs are instantiated inside a global EORI database as EOR instances. All EORI attributes are assigned values. Also ProSAps may instantiate EORMs into their proprietary PPR models.
~ IIM and MTRT and EOIs and EORIs form the GIS information contents and are accessible via GIS-wide services.

The proposed handling of specific knowledge and abstract background knowledge within the GIS facilitates a *high-level information flow*, as it has been requested above. The modeling of engineering object relations as equally important entities promotes a *flexible, clear and structured representation* matching also the feature philosophy of attaching information to exactly where it refers to, thus avoiding unintelligible all-in-one and hard-to-maintain knowledge bases.

9.3.5 Meta-Information

The specification of explicit semantics and context information has already been identified as relevant meta-information, which is to be attached to informational entities. This section details them and covers other relevant issues.

Context and Semantics. It has already been argued that the definition of IEs, including their semantics, is always context-specific and that contexts are a method to manage multiple different views on information pools, which in turn allows for view- and task-oriented optimization of IEs.

Ostermayer emphasizes in *[Ostermayer, 2001]* the importance of covering the pragmatic and situative aspects of information by representing context information explicitly, when informational entities are taken out of their embedding within single process step models and joined with other process steps' information (see also section Part III – 7.1.3). Ostermayer suggests that such context information has to describe an IE's intended usage, as each different purpose of usage may require to represent different aspects of the same IE (in Ostermayer's approach, same IEs are identified by a common identifier). From Ostermayer's findings, it is concluded here that each informational entity should be attached with information about the context in which it is valid, and that the context information itself should – amongst others – express the purposed usage of the IE. First details have already been developed in section Part III – 7.1.3 above.

Therefore, each informational entity (EOx, especially EOR types) within the GIS has to carry explicit meta-information on its semantics, as well as context meta-information, expressing the scope within which this EOx is valid with the given semantics. Instance information gets its semantics from the respective abstract information, but has to carry context information, too.

☞ Many of the conclusions drawn in this section's discussions will be mirrored and comprehensively made visible in the representation formalism developed for ULEO (see the section *Information Representation Formalism* below).

Context

From what has been explained in the preceding section, it can be concluded that any specific EOx (including instances) can be part of **multiple contexts**; different types of EORs can be used to correlate them. Additionally, there might be any number of EOx carrying the same identifier but different context information. Thus, context information of an IE is represented on **two levels**: indirectly (implicitly) by the IE's environment consisting of other IEs, and directly by its explicit context specification. As a result, also the IE's environment may be divided up into several different environments, each of which is valid in a certain (explicit) context. Thus, explicit context specification is the first step in retrieving an IE's context, functioning as a kind of filter, after whose application the indirect context can be retrieved by considering the IE's environment.

Thus, the global information space, GIS, divides itself up into possibly many partitions, possibly overlapping and being related to and accessible to each other by relationships and/or common identifiers. This supports varying usage and meaning of EOx along the different domains in product development. Conflicting information may be explicitly represented and handled (keyword *transparency*). Furthermore, it becomes possible to find all kinds of information relevant in a given context, including unfinished information currently being under edition. Thus, if users or applications want to store new background knowledge, they will be able to use UMEO as central entrance to find out, if others are already doing the same. This is, for example, relevant when aiming at storing new automation strategies. Relevant information will be retrievable by specifying EOx IDs, context and semantics descriptions.

✎ To use a picture, if a single context is thought of as a two-dimensional network of nodes and threads, the result will be a possibly densely interwoven piece of cloth; the elements within the cloth will be any kind of EOx: EOx are the nodes, EORx are the threads between nodes.

The cloth is multi-layered, and one context is one layer in it. The individual cloth layers might be everything between not interwoven at all and highly interwoven – by common EOx (shared between contexts) or just by common identifiers (there might also be cases where two EOx sharing the same identifier, will have different properties within different contexts).

A ***table of identifiers*** can be generated and managed above all these cloth layers. An individual identifier plus EOx type may refer to all occurrences inside the cloth. That way it is possible to detect cases of synonyms and homonyms:

~ Synonyms: EOx with the same meaning might have different identifiers in different contexts. The degree of similarity that has to be reached for speaking about synonym EOx has to be specified by the retrieving process. Similarity depends on the set of attributes (types), the set of methods (types), and the set of relationships (here, limitations for following* relations are necessary).

~ Homonyms: the same IDs are used for EOx with different meaning. This is valid also within the same context, as the addressing schema (see section *Identification and Addressing Schema* on page 122) specifies the name scope. This is not valid within the same name scope (e.g., all subclasses of a parent class).

The exact way of representing (explicit) context information depends on the kind and limits of the global information space that is in focus, i.e., on the implicit assumptions made for the information handled within a given information space. As adding the same portion of context information to all IEs produces unnecessary overhead and redundancy, it is suggested to keep the specification of context schemas **flexible** by agreeing upon a common minimum kernel and allowing any additional kind of context information to be added. A common minimum kernel, suggested in the following as a first choice proposal, should contain such information necessary for a working information sharing (see also section Part III – 7.1.3 above for a first discussion):

An **explicit context information schema** must consist of a **domain identifier**† such as *detail design*, and *machining planning* – describing the IE's purposed usage – plus a domain-specific **locator string**, specifying the storage location within this domain's information space up to a degree of detail, where EOx identifiers are locally unique. As already pointed out above, in cases where several applications are used within the same domain, and where these applications use different information organization structures, it is sensible to specify the application name‡, too, for being able to handle all cases. For the same reason, introduction of a freely structurable *further context* is proposed.

☞ As a result, a globally unique EOx address is used instead of a single identifier. Without context information, globally unique identification rules would be required (see above). Dough Lenat describes in *[Lenat, 1998]* a more detailed representation of context information aiming at building up a database for as much knowledge about the world as possible. It has to be pointed out that the locator string may vary from domain to domain, according to the individual ProSAps' organization of information (see below for examples). Note that referring to the picture of a multi-layered cloth, the *domain* plus the optionally filled *further context* information are the ones that will most probably separate the layers. However, depending on the domain and the complexity of information organization structures of an application, it may also be sensible to additionally use part of or the complete locator information for identifying such major contexts, which arrange informational entities into groups of practical meaning, reflecting a portion of the product development process.

Footnotes

* In the sense of virtually walking along

† Or some equivalent to it

‡ As non-mandatory information

Identification and Addressing Schema in the GIS

What has been developed so far for achieving a global information space allows for free flow of high-level information. To achieve transparency of information in the sense of finding everything, possibly of relevance, it has already been stated that all informational entities inside the GIS have to be augmented with information about the scope within which they are valid, i.e., the **context**. Detailing this idea, an **identification and addressing schema** (see Figure 22 and Figure 23) must ensure that informational entities can be stored and retrieved without ambiguity. For this purpose, **unique EOx addresses** have to be used as combinations of locally unique EOx identifiers plus context information. The alternative, globally unique EOx identifiers, would be incompatible to the idea of using de-central handling of EO instances, as the latter includes de-central assignment of EO instance identifiers.

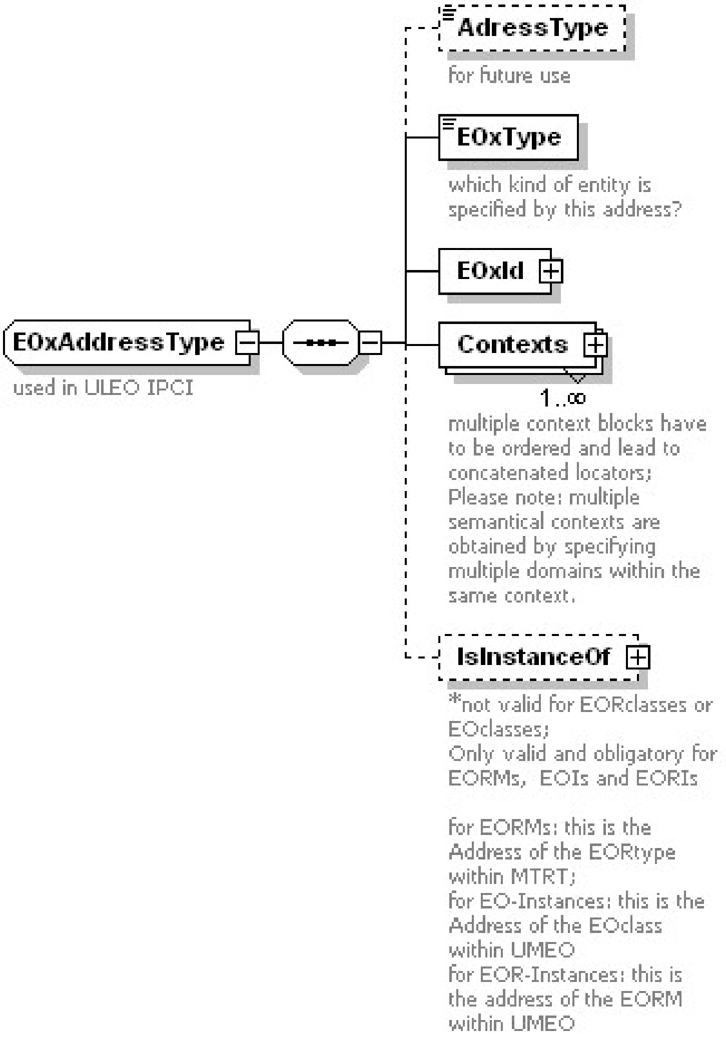

AdressType
for future use
EOxType
which kind of entity is specified by this address?
EOxId
EOxAddressType
used in ULEO IPCI
Contexts
1..∞
multiple context blocks have to be ordered and lead to concatenated locators;
Please note: multiple semantical contexts are obtained by specifying multiple domains within the same context.
IsInstanceOf
*not valid for EORclasses or EOclasses;
Only valid and obligatory for EORMs, EOIs and EORIs
for EORMs: this is the Address of the EORtype within MTRT;
for EO-Instances: this is the Address of the EOclass within UMEO
for EOR-Instances: this is the address of the EORM within UMEO

Figure 22: EOxAddress Schema, ULEO XML

Using this kind of addressing allows ProSAps to retrieve relevant information in many cases without having to know the exact application's name, which is desirable within an open information space with a varying set of participating applications. As a mapping of context to applications has to be done somewhere, this method suggests the existence of a central application working as a directory listing applications offering services. In other words, this central application performs address resolution. It will be designated as an **EOx address resolver** (see the section *EOx Address Resolution* below).

☞ In cases where multiple applications are used within the same domain, fully application-neutral information handling only works, if the information within this domain is managed according to the same structure by all respective applications. In addition, this case requires that the addressed application be specified as part of the context.

Multiple contexts. As has been argued above, each informational entity can be assigned with more than one context structure, each of which may contain more than one domain and/or application name. If more than one context structure exists, the respective IE can be accessed (retrieved or stored) by sequentially resolving all of them. One example for this is the retrieval of an individual feature instance from a PDM system, where in the first step the CAD file is retrieved from the PDM system (context 1), and in the second step the feature instance within the CAD file is retrieved (context 2).

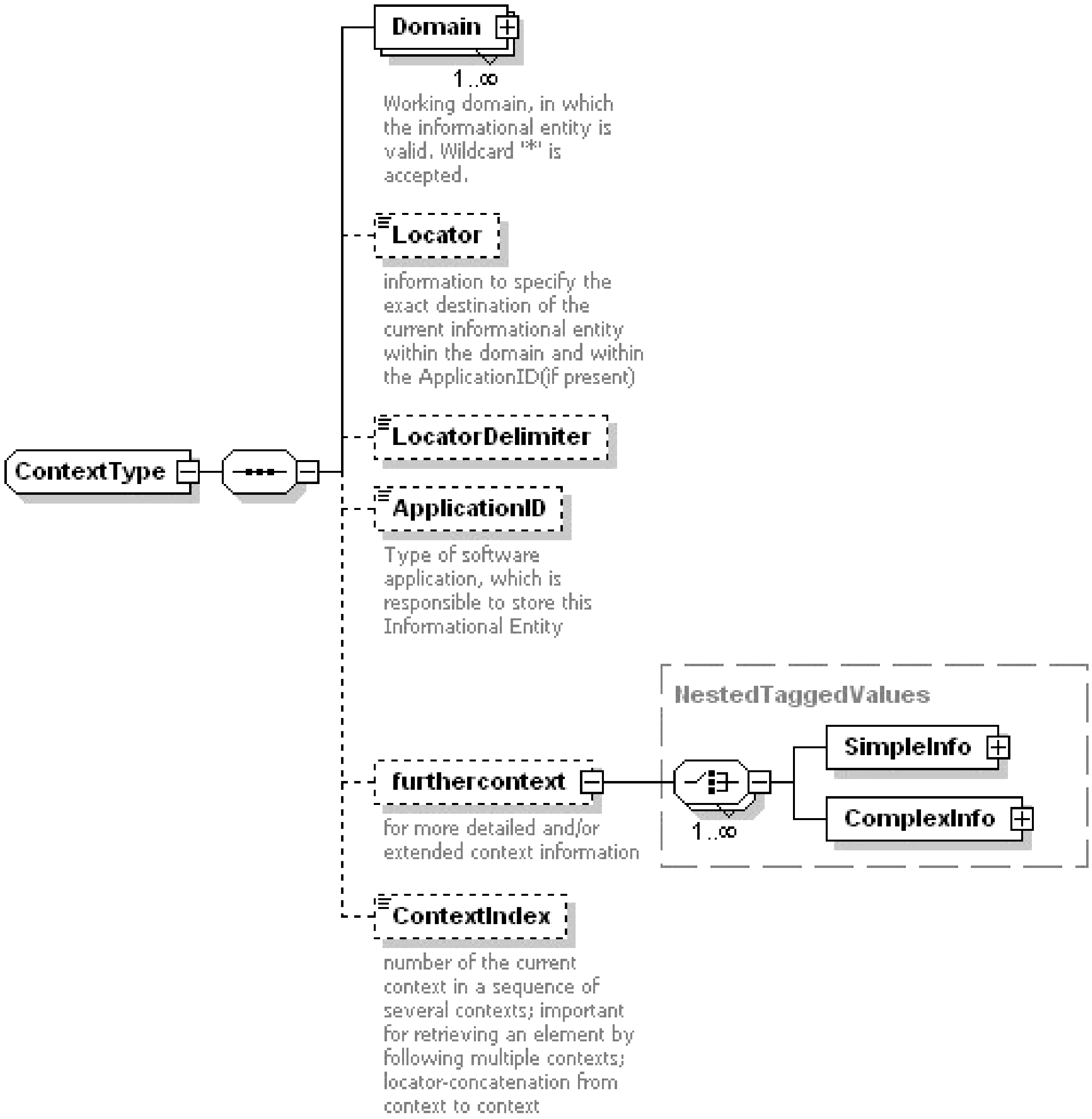

Domain
1..∞
Working domain, in which the informational entity is valid. Wildcard "*" is accepted.
Locator
information to specify the exact destination of the current informational entity within the domain and within the ApplicationID(if present)
LocatorDelimiter
ContextType
ApplicationID
Type of software application, which is responsible to store this Informational Entity
NestedTaggedValues
SimpleInfo
furthercontext
1..∞
ComplexInfo
for more detailed and/or extended context information
ContextIndex
number of the current context in a sequence of several contexts; important for retrieving an element by following multiple contexts; locator-concatenation from context to context

Figure 23: Context Specification in ULEO XML

Semantics

It has already been argued that a ProSAp can only process another ProSAp’s informational entities correctly, if it knows their semantics (defined by the sending ProSAp). Expressing semantics is not only important for EO classes, but is especially crucial for relations, as has been shown above.

✍ To shortly recall: applications can at run-time select the best-fitting relation types from MTRT before following the respective EORMs or EORIs in order to retrieve the desired information. Compare also for the example of PDM systems, being able to identify CAD models affected by the changes in another CAD model having been updated.

Thus, direct semantic descriptions should be attached to any EOR type within the MTRT and to any EO class within IIM/UMEO. Additionally, semantics has been shown to be context-dependent. Since an IE’s context results from explicit specification and from the IE’s environment, also the IE’s semantics depends on these factors. And, according to the context specification, also **semantics** can be (partly) expressed **explicitly** within the informational entity. This issue will be handled within this section.

✍ To recall the recipients of explicit semantic descriptions: as already discussed, semantics has to be understandable for human beings developing IT for Engineering or consuming information. However, the more semantics is additionally also representable in a machine-interpretable format, the more flexible become ProSAps in using most-possible information inside the global information space.

In order to assure readability and comprehensibility for human beings and to assure the covering of a sufficient portion of the full semantics, an explicit semantics description must comprise a section of **natural language** and/or other common languages for describing semantics such as predicate logic. Since the human information consumers consist of IT experts and end users, it is suggested to consider formal but also informal semantics descriptions.

☞ Other relevant information for the end user is represented within attributes. One of the most prominent examples is the *function* attribute within design features[*]. There is not always a distinct borderline between semantic description and other attributes of an EOx.

For supporting machine-interpretation of explicit semantic representations, one can either use a formal **description language for representing semantics**, or use pre-defined propositions, forming a **description schema**, commonly agreed upon by all participants within certain contexts – as with contexts, also the description of semantics may be based on top of many or few presuppositions, implicitly made by the information producers and consumers; it may simplify the explicit semantic descriptions, if many presuppositions are made. To achieve a maximum exchange of information, however, the description of the semantics must be as generally applicable as possible. To handle both, semantic descriptions can contain *both* universal and context-specific elements.

✍ If a formal **description language** is used, grammatical descriptions including syntactic and semantic rules, must be centrally documented, too, in order to allow development of software algorithms to interpret it. For the same reasons, specified below for KR languages (section *Information Representation Formalism*), grammar of such description language(s) is suggested to be expressed using natural language. In order to achieve maximum information availability, the central documentation is proposed to be stored in a central ***table of semantic description***

Footnotes

[*] For example, saying “I am a sealing feature.”

languages (TOSDL), within which each language (preferably there is just one) is assigned a unique identifier.

☞ A detailed definition of a description language or a semantic schema is considered to be beyond of the scope of this research, mainly due to the restriction of resources.

A **schema for the explicit description** of an informational entity's **semantics** could comprise the following elements:

(1) Natural language semantics for end users (i.e., engineers)
  a) For EOR types
  b) For EO classes
(2) Natural language semantics for IT experts
(3) Formal semantics for IT experts (predicate logic)
(4) Description language
  a) Formal semantics via description language
  b) Unique identifier of used description language
(5) Schematic semantic description
  a) For EOR types
    a-1) Transitivity: yes/no
    a-2) Inheritance type
  b) For EO classes
    …

The meaning of the allowed values for the individual schema elements must also be commonly agreed upon: sets of values, number intervals, and single numbers.

Other Meta-Information

The current sub-section briefly discusses further sorts of meta-information, possibly relevant in practical applications.

Depending on a company's IT processes, some or all of the following sorts of meta-information may be of relevance. However, due to the same dependency, this list cannot be regarded as complete; thus it is suitable to keep meta-information flexibly adjustable according to those needs. The subsequently discussed issues shall provide an impression of the variety of meta-information.

✎ All types of meta-information discussed below are applicable for all kinds of informational entities.

Information on **author**(s) and modification facilitates tracing an IE's journey through life. **Versioning** information seems of special relevance, since it raises some general questions such as: what happens to EO(R) instances, whose respective classes have changed? For a discussion of those issues please refer to the section *New Processes* below. The respective meta-informational attributes are in short:

~ Created by
~ Created at
~ Last modified by
~ Last modified at
~ Change history
~ (Class) version ID

Handling information always touches issues of **data security**, i.e., protection of company's and personal intellectual properties*. As a basis for facilitating security management, each IE can be equipped with company- or context-specific access rights, confidence or licensing information, or can even be assigned a use-by date, after which the IE will be destroyed by the application using it. The respective meta-informational attributes are in short:

~ Access rights (if necessary, specific for user / group x)
~ Confidence state
~ Required licenses
~ Date of self-destruction

From current product development's view, rather unusual possibilities are presented by handling **modality** and **reliability** / **(un-)certainty** of information. These issues will be subject to future research. Figure 24 displays the universally applicable representation for meta-information within ULEO XML.

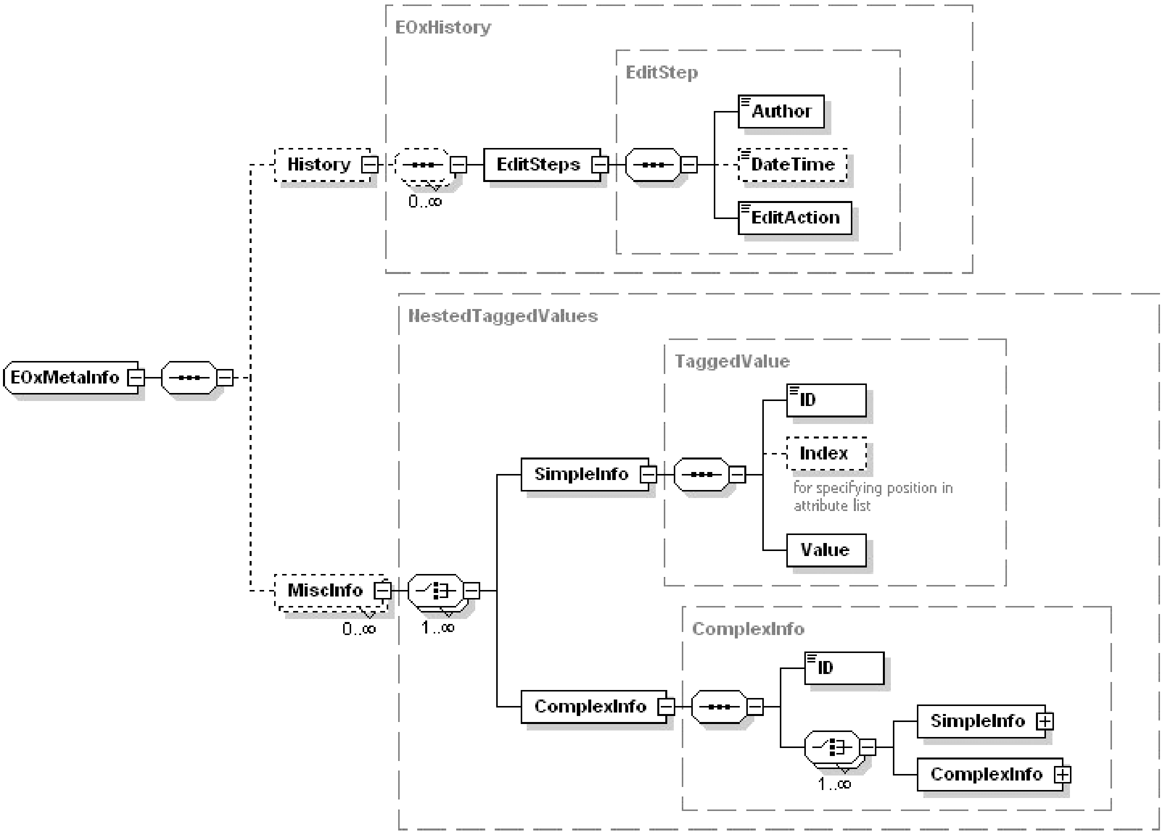


Figure 24: Universal Meta-information Schema Within ULEO XML

The section *Standard Representation Formalisms* below will state that also representation formalisms of IEs' and their attributes and methods are a relevant sort of meta-information.

Footnotes

* It is assumed that physical data integrity is maintained by the underlying IT systems; therefore, measures such as checksums are not considered here.

9.3.6 Realizing a Global Information Space

During the preceding parts of section 9.3, a method for structuring information has been suggested. In order to allow different sorts of applications to utilize and provide such information within a global information space, where all needed information can flow freely amongst all participants, further measures have to be taken.

Inter-Process communication

There already exist powerful and ready-to-use technologies for inter-process communication (IPC) such as CORBA and Web Services. Available IPC functionality can be added to existing ProSAps by customization (see above). Thus, it is necessary to agree on one or more technical solutions, which will be done during section *Part V – 11.2.1* of this thesis.

Current inter-process communication methods are service-based. This means that they offer a set of **services** each that can be used by interested clients by calling – from within their source code – methods provided by respective infra structural software. These service calls will be routed to the desired service provider that finally executes them and eventually returns some result information.

Services in this sense can be used to exchange information. As in principle any application is allowed to offer and to use any service, this is a good basis for implementing a global information space.

Information Representation Formalism

Text-based representation. Of course, it is not enough to specify a set of services and a base technology for inter-process communication in the GIS: in addition, commitments have to be made on how information adhering to the above-specified information structure will be exchanged via IPC; it has to be agreed on a **representation formalism** for the user data. More generally, from what has already been discussed, it can also be concluded that there is a need for a machine-interpretable formalism to represent information within UMEO and MTRT and to exchange or share it amongst applications in the GIS, i.e., a common formalism to carry varying contents of information models as well as specific (instance) information and meta-information.

For maximizing openness of the global information space, neutral **text** formats are useful. Although they require more storage room as binary formats do, this benefit should be ranked more important, as openness is critical to include as many ProSAps as possible into the GIS. In particular, the discussion of the state of the art, which is condensed in Part III – Chapter 8 above, suggested using **XML** as the basis for a new representation format, due to the fact that representation formalisms on XML basis provide useful pre-structuring and allow the use of a range of standard tools. For detailing and fixing the most suitable XML format, also called *GIS representation format* or *ULEO XML format* in the following, please refer to section *GIS Representation Format ULEO XML* below.

Contents of attributes and methods. For the given reasons, also the contents of attributes and methods must be text-based. However, it has not been stated, which formalism should be used for this purpose, as the GIS representation format, reflecting the GIS structure, merely specifies the frame for transporting this information. In order to allow for storing as manifold information as possible by rising expressive power, it is suggested not to restrict the kind and number of representation formalisms allowed to be used within attributes and methods. This suggests in turn introducing a ***table of information representation formalisms (TOIRF)***[*], listing the formalisms' unique names and descriptions (a language's grammar etc.); doing this, the *meta-information* should be attached to each

Footnotes

[*] The notion of knowledge representation is more common than that of information representation.

informational entity, specifying which formalism is actually used within in. This leads to a **hybrid representation** of attribute and method contents within the GIS representation format.

✍ The grammar of the representation formalisms used within attributes and methods has to be understandable for human beings developing IT for Engineering[*] or consuming information as end users. In principle, the more syntax and semantics are additionally representable in a machine-interpretable format, the more flexible become ProSAps in using most-possible information inside the global information space. Machine-interpretable grammar descriptions have to be represented in a set of formalisms[†] (see *[Haugeneder & Trost, 1993]*), commonly agreed upon within IT for Engineering. However, due to the complexity of these issues, it is suggested here to represent grammar in a first step for human recipients only using natural language texts. Because (1) this solution will cover the needs of the presumably most common case of hard-coding language interpretation within ProSAps, and (2) since the exploration of suitable software-interpretable formalisms is a large field of its own, it is not considered to be in the scope of this work.

Details on Hybrid Information Representation within Attributes and Methods

Hybrid representations generally allow for representing information more naturally, as there is no single information representation formalism that is assumed to be ideally suited for all kinds of information. As it seems not predictable, which representation formalisms will be suitable for possible future kinds of information, open hybrid representation seems to be necessary for reasons of assuring future usability of IT concepts.

☆ For example, Fuzzy Logic may be intuitive for certain domains in the future, and inspection strategies can be naturally represented by a dedicated and optimized language. In *[Zimmermann, 1994]*, the author integrated a terminological knowledge representation formalisms (*SB-ONE*, which is based on *KL-ONE*) and a solver application working with first-order logics (*OTTER*); this is a good example that even the commonly agreed very high expressivity of the predicate calculus is not enough in any case – especially in terms of adequateness and naturalness of representation.

This promotes usability, which is one of the requirements placed on future IT for Engineering (see catalog). In principle, a hybrid representation also opens the global information space for a larger variety of ProSAps[‡]. From the decision to manage instance information proprietarily arises the need for hybrid representation of attribute- and method contents, too, as various systems may store their attribute values using different delimiters for decimal points or proprietary formulas. Thus, the table of information representation formalisms will contain at least the different systems' languages names, allowing information recipients to adapt to that and to convert values if needed.

☞ Hybrid representation as proposed here merely targets contents of EOx attributes and methods. Permitting hybrid representation also for the GIS basic structures (whole GIS objects) would lead to a much less pre-structurable information space and thus make the common and open cooperation framework less efficient and powerful.

To shortly sum up the last propositions: It is suggested to introduce a GIS-wide, XML-based representation formalism, commonly agreed upon by all participants of the GIS and reflecting the GIS's basic structures. In more detail, this formalism has to be able to express EO classes and

Footnotes

[*] See also the paragraph on semantics within the section concerning requirements for future product development tools

[†] Such as *Backus-Naur* for syntax and predicate logics for semantics.

[‡] Catalog keyword: *openness to any application and information source*

instances, EOR types, EORMs and EORIs, together with the respective meta-information. The contents of EOx attributes and methods should be representable in any text-based representation language. However, some commitments are useful, as the next section *Standard Representation Formalisms* will show.

Standard Representation Formalisms

On top of what has been discussed on hybrid representation, it is suggested for the sake of easing communication in practical use, to additionally introduce a basic set of **standard atomic data types** (also called basic data types in this thesis) and a suitable ***Standard Information Representation Formalism, (SIRF)*** to cover strings, floating (real) numbers, integers and similar. Generally, data types can be represented within the IIM analogous to EO classes, but attached with a respective meta-information designating them as data types.

☞ Two suitable sets of data types can be taken from the representation formalisms *EXPRESS* (see STEP) or *XML schema*. The decision about a suitable SIRF has to be made GIS-wide and is thus due to the company, managing the GIS. However, it is proposed here, to generally adopt XML schema (see *[W3C XMLschema, 2001]*) for providing data types and representation format SIRF, which are widely spread, commonly accessible and compatible to XML and the XML-based Web Services inter-process communication.

☞ Please note that the selection of a SIRF as defined above applies only for basic data types.

Inside the respective EO(R) class meta-description, the information about the use of this SIRF must be specified. In order to keep the option to transport proprietary expressions within attributes (e.g., formulas), it is suggested to allow EO(R) instances to overwrite that default language specification by the respective proprietary one in such cases.

On top of what has just been suggested, it is still possible to have applications stick to the SIRF only, thus having to convert information contents accordingly.

✍ On top of the hybrid representation, there is, in principle, still room for even more variance in representation languages in respect to the languages eventually offered to the user to enter information[*]: applications are of course free to convert such IEs' contents to other formats that are used for final storage.

✍ In addition to symbolic representation formalisms, also **sub-symbolic** ones such as neural networks are conceivable to bring benefit in certain areas of the domain. However, in most cases, symbolic formalisms will be preferred because of their common comprehensibility and the traceability of reasoning.

✍ It is conceivable to allow more than one representation formalism inside the same IE.

To again sum up the last propositions, briefly: ULEO standardizes the GIS information structure down to the detail of attributes and methods. Attributes' atomic types and contents may be represented based on a proposed minimum set of standard formats or using any newly introduced format. It is suggested, to adopt XML schema basic types for this purpose. It is important to point out that the sets[†] of attributes and methods of individual EO- and EOR classes, however, are non-standardized, which yields the most important effects: there is still the flexibility of representing any desired proprietary concept and relation type. Meta-information and typed relations permit such usage of non-

Footnotes

[*] Such as expert knowledge

[†] In the sense of „collection“ or „constellation“

standardized informational elements as well. Each informational entity has to carry meta-information on its semantics, its contexts, and the representation formalisms used within it. The latter is true also for attributes and methods.

Local and Global Consistency of Information in the GIS

It must be pointed out that **global consistency** of the information within the GIS cannot practically be assured when using hybrid representation. Non-standardized informational entities (EO classes and relations) lead to the same effect, as semantics will presumably never be fully machine-interpretable, so there will be no machine-maintained global consistency of information in the global information space. However, it is argued here that this is not desirable, anyhow, as, for example, also unfinished and contradictory information shall consciously be stored within IIM/UMEO – UMEO is intended to serve as a central entrance point to find and access all relevant information – this includes also contradictory propositions about the same matters. Furthermore, it has been claimed to tackle the "chaos of the real world"; however, the real world is inconsistent in itself. Thus, it is recommended that consistency is maintained locally where it is really needed instead of introducing a central mechanism trying to assure global consistency. "Locally" means "in a given (set of) context(s)", it does not mean to identify a certain set of IEs, as they may belong to several contexts, containing contradicting information.

Semantics of Representation Formalisms

After some general thoughts on semantics within representation formalisms, a concrete formal semantics for the GIS representation format will be proposed.

How much meaning should a representation formalism transport?

As argued above, a representation formalism should possess a **formal semantics** for being machine-processable. Thus, the formalisms should know general basic information types such as abstract and instantiated concepts and relations, and respective kinds of meta-information.

✍ What can an application do with the **formal semantics**? If an application is able to syntactically distinguish the representational elements of an representation formalism (i.e., by parsing it), it can apply hard-coded algorithms on them, considering the respective semantics; for example, concept types can be treated differently from relation instances, each by a standard procedure – this includes some fundamental semantic consistency and completeness[*] checks, but may also cover the processing of inheritance of attributes and relations between them. Thus, the processing application won't be able to perform miracles, but a good set of basic functionality can be provided.

Nevertheless, in principle, the semantic content carried by a representation formalism cannot be sufficient for integrating applications, as a language must not know and thus prescribe the individual types of entities to be shared by the applications; in this case it would be too specialized for being able to handle flexible information contents; representation formalisms working with a fixed set of concept types and relations do provide a high degree of semantics to the using applications, but are too rigid. Consequently, a formalism should also provide means to **explicitly represent** context-specific **semantics** by built-in structures, and to **implicitly** represent it by **powerful relations**, and, finally, to

Footnotes

[*] For example, does an instance have all its attributes?

inherit it between entities. Implicit semantic representation supports synthesizing an IE's semantics by looking at its relational network to other entities.

A complete description of an IE's semantics using representation formalisms is impossible, by nature; this is true for any kind of representation formalism, no matter, whether it is machine-processable or not – even for natural language, which is still the most powerful formalism. Because of this, in addition to applying a sophisticated representation formalism as just described, there should exist sets of IE's in the GIS, serving as ***semantic kernels*** (see section 9.2.1 above). It has already been argued that commonly agreed **relation types** are powerful means of supporting integration starting off from such semantic kernels, if the representation and the applications are geared for that purpose.

What can an application do with **semantic kernels plus** "gluing" **relations**? As already discussed for semantic kernels, in the ideal case of a sufficient semantic exploration, each type of concept and relation can be individually processed in a way, adapted to its engineering meaning.

Formal Semantics of the ULEO GIS Information Structure

Definition 1: A ***GIS information structure*** is a structure consisting of the disjoint sets ***SEOC***, and ***SEORT***, and ***SEORM***, and ***SEOI***, and ***SEORI***, and the partial orders $\leq_{SEOC}$ on ***SEOC***, and $\leq_{SEORT}$ on ***SEORT***, and a function f_{EORmat} : ***SEORT*** → ***SEORM***, and a function $f_{EORinst}$: ***SEORM*** → ***SEORI***, and a function f_{EOinst} : ***SEOC*** → ***SEOI*** …

$$\textbf{\textit{GIS information structure}} := (\ \textbf{\textit{SEOC}}, \leq_{SEOC}, \textbf{\textit{SEORT}}, \leq_{SEORT}, \textbf{\textit{SEORM}}, \textbf{\textit{SEOI}}, \textbf{\textit{SEORI}}, f_{EORmat}, f_{EORinst}, f_{EOinst}) \qquad \text{(I)}$$

…where the set ***SEOC*** is called *set of Engineering Object Classes*, while its elements are called *Engineering Object class representation*s, or in short *EO classes,*

…and where the set ***SEORT*** is called *set of Engineering Object Relation Types*, while its elements are called *Engineering Object relation type representations*, or in short *EOR types,*

…and where the set ***SEORM*** is called *set of Engineering Object Relation Materializations*, while its elements are called *Engineering Object relation materialization representations*, or in short *EOR materializations (EORMs)*, and each of which is a materialization of exactly one element of ***SEORT***,

…and where the set ***SEOI*** is called *set of Engineering Object Instances*, while its elements are called *Engineering Object instance representations*, or in short *EO instances (EOIs)*, and each of which is an instance of exactly one element of ***SEOC***,

…and where the set ***SEORI*** is called set of *Engineering Object Relation instances*, while its elements are called *Engineering Object Relation Instance representations,* or in short *EOR instances (EORIs)*, and each of which is an instance of exactly one element of ***SEORM***,

…and where the partial order $\leq_{SEOC}$ on ***SEOC*** is called *taxonomy of EO classes*, where $\text{EOclass}_1 \leq_{SEOC} \text{EOclass}_2$ implies that EOclass_2 generalizes EOclass_1 (subsumption)

…and where the partial order $\leq_{SEORT}$ on ***SEORT*** is called *Meta-Taxonomy of Relations Types **(MTRT)***, where $\text{EORT}_1 \leq_{SEOC} \text{EORT}_2$ implies that that EORT_2 generalizes EORT_1 (subsumption)

…and where the function f_{EORmat} is called *EOR type materialization,*

…and where the function $\boldsymbol{f_{EORinst}}$ is called *EORM instantiation*,

…and where the function $\boldsymbol{f_{EOinst}}$ is called *EO class instantiation*.

Definition 2: For reasons of clarity, it is informally defined that …
…relations can correlate any number n ≥ 1 of concepts and/or relations,
…EO classes and EORMs may inherit the involvement into relations from their subsuming types (EO classes or EORMs of a parent type), such that for any EOR type in MTRT it is defined, if and how it will be inherited through the inheritance relations in the IIM and in the MTRT.
…multiple inheritance is possible.

Definition 3: An ***Integrated Information Model (IIM)*** is a structure ***IIM*** consisting of the disjoint sets ***SEOC***, and ***SEORM***, and the partial orders $\leq_{SEOC}$ on ***SEOC***, as defined in (I)

$$\boldsymbol{IIM} := (\boldsymbol{SEOC}, \leq_{SEOC}, \boldsymbol{SEORM}) \qquad \text{(II)}$$

Thus, ***IIM*** is a sub-structure of the ***GIS information structure***.

Brief Summary

To sum up, it is suggested that an **XML-based GIS representation formalism** be employed on top of a **standard IPC method** and an **EOx identification and addressing schema** to facilitate a sophisticated, GIS-wide, non-proprietary communication protocol to be used by ProSAps and central services. The GIS representation format has to reflect the above-suggested information structure by offering means for expressing the following informational base types: concept- and relation classes (abstract background information), concept- and relation instances (specific information), and meta-information. Thus, all the information accessible through this service-based interface is clearly structured yet sufficiently flexible to carry any desired type of concept or relation.

It has also to be pointed out that the approach proposed is contingent on the existence of persistent (not necessarily unique) identifiers of EO(R) instances within ProSAps, as this is a prerequisite to be able to refer to them reliably.

9.4 Automation

Facilitation of automation in product development is another goal of this research. This section discusses details.

Automation of tasks invariably calls for input information, i.e., source information to be automatically processed and produces output in terms of action triggers (control information) or user information, while using some automation information (knowledge).

In pure product development, such input and output will always be information. If information is structured according to the ULEO approach, input and output of automation will comprise EO classes, EORMs, EO instances, and EOR instances, resembling classes and instances of concepts and relations. Automation of product development tasks in the strict sense will always result in an automated instantiation or change or deletion of EO(R) instances inside ProSAps' product models. As automation knowledge is applied during the automation, and as automation transforms input- to output information, the **modeling** of **automation knowledge** using **EOR(M)s** suggests itself as a natural solution. Consequently, EOR instances will correlate the respective input and output instances and document the kind of automation knowledge applied (by documenting the EORM containing it).

✍ To resume this in some more detail, EORx can correlate a single application's EOx but also bridge multiple applications, which is, for example, the case with *feature mapping*. Automation knowledge is by nature formulated generally and thus applicable for EO classes and EORMs (i.e., abstract information). Thus it is part of IIM/UMEO. After its application, input entities and output entities can be tracked, together with the applied knowledge entities (EORMs), by linking the relevant EO(R) instances through an instance of the applied EORM. The application of automation knowledge stored inside IIM/UMEO is hence reflected in the PPR models and the EORI database. This approach meets the request for representing mapping knowledge inside a kind of conceptual relations, as formulized while discussing the feature linking approaches in the state of the art.

9.4.1 Generative EORs

This section works out a special kind of relationships dedicated to the support of automation.

Scalability and flexibility have been identified as major criteria for influencing any IT concept's chances of being adopted in practice. Automation is especially critical from this point of view, as it is suited to change current work processes significantly and thus to provoke the resistance of the people involved. It hence seems worthwhile to provide further means of scaling the degree of automation without having to modify the newly introduced software. One way to achieve this is to add information to UMEO detailing which elements of the automation knowledge are to be processed in which situations.

✍ In this case, UMEO, and not IIM as a whole, seems to be the appropriate place for storing such information, as otherwise it would be too distributed and consequently difficult to retrieve efficiently.

This idea leads to a differentiation of EORs into ***Informational EORs (IEORs)*** on the one hand and ***Generative EORs (GEORs)*** on the other. IEORs represent background information to be used by ProSAps (e.g., ontological information and automation knowledge), whereas GEORs represent generative information. Both kinds of information are integral parts of UMEO.

✍ **Some motivation on the terminology** used here. Informational EORs are part of the very core of the user information / knowledge. In other words, they literally serve to inform users and applications. Generative EORs, on the other hand, are not supposed to inform ProSAps but should instead control the details of any automation performed by them. They are generative in the sense that they generate automation results by triggering and controlling the EO(R) instantiation.

Details on IEORs. The preceding sections have discussed what are now called *informational EORs* or IEORs: within IIM/UMEO, they describe logical relationships between EO classes and/or EORMs, e.g., *is_inspected_by* or a relation between *n* design features and *m* machining features, which might be called *is_machined_as* and could describe the machining features that are necessary to machine the related design features. Thus, IEORs represent what is called **engineering knowledge**. Moreover, with IEORs, UMEO and the IIM as a whole may be augmented gradually by ontological information, thus adding sophisticated information about the EOs' semantics, usage, and embedding within the extended contexts of other applications' information. More generally expressed, this is background knowledge about the world. **IEOR instances** inter-relate individual PPR models by linking the logically corresponding EO instances inside them, permitting information to be interchanged between ProSAps on the level of features and other EOs. Multi-directional associativity (intra-domain and inter-domain), including horizontal and vertical associativity as discussed during the investigation of product engineering's current state, can be facilitated using IEORs. Hence, IEORs have the potential to simplify informational integration of the applications along the process chain. They may be utilized by application programs for a wide variety of purposes such as Design-for-X (for example, cost estimation, manufacturability checks) or feedback from downstream to upstream applications.

Details on GEORs. Generative EORs represent information on how to use domain knowledge[*] automatically. The result will be, as stated at the beginning of this section, the instantiation of EOx instances being parts of product, process, or resource models, although it might also result in entering new abstract information into UMEO. Such GEORs that result in instantiating EO classes and EORMs automatically are triggered by preceding instantiations and are a subtype of GEORs – they will be called AutoCreate relations or, in short, ***AutoCreates (ACs)*** in the following[†].

☆ For example, a set of machining planning features can be generated as output based on a set of finish-part features serving as input. Analogously, measuring elements constituting an inspection plan can be generated according to tolerance objects (see Figure 25).

Automatic instantiation may occur triggered by and taking into account conditions inside PPR models, thus providing **EO mapping functionality**. Another example is **re-instantiation** of EO instances, i.e., existing EO instances are commuted into instances of other EO classes.

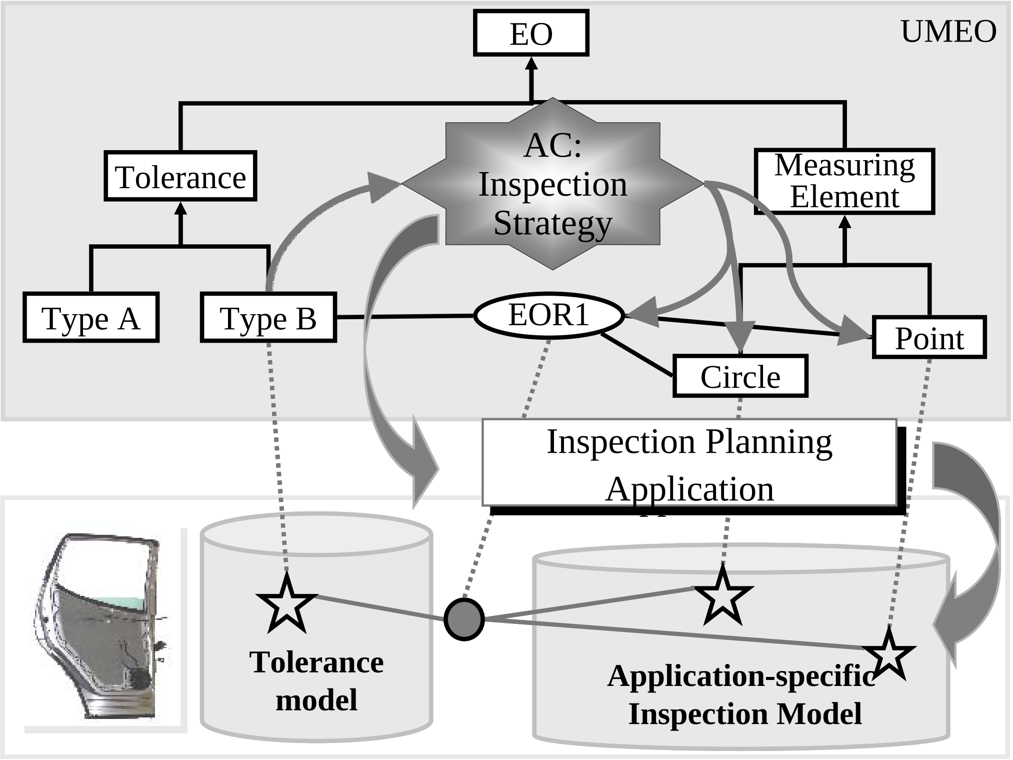


Figure 25: Example of an AutoCreate Relation in the Quality Assurance Domain

Although GEORs resemble a kind of meta-information, they will be represented using the standard EOR structures and stored within UMEO in order to assure efficient processability. The alternative solution of managing GEORs within a separate meta-model would yield opposite results.

Footnotes

[*] IEORMs and EO classes

[†] Other GEOR types might handle insertion of EO classes into UMEO.

GEOR contents are often of a procedural nature and could also be seen as **micro-workflow containers**. Examples are instantiation procedures for features and their attribute values. The interpretation of GEORs results in an automation of engineering tasks. GEORs provide a functionality subsuming **feature mapping**; yet they are, in contrast, not limited to feature classes – in fact, they may consider any kind of engineering object. After each manual modification of a specific PPR model, a linking algorithm may locate the EO class of the manipulated EO instance within UMEO and search for GEORs related to it. Then, the GEORs' contents are interpreted by the system. For example, these contents could define values of feature attributes or the number of new instances to be created.

Scalability and **flexibility**, as requested above, are achieved by adding or removing GEORs to UMEO or by changing their contents. For example, new applications may be rolled out without offering automation functionality. Then, after users have been trained and any teething troubles have been remedied, an increasing number of automation tasks are gradually performed. The finally selected degree of automation is not predictable in all cases with technological changes being a possible reason. Nevertheless, GEORs enable the degree of automation to be raised – also to initially unexpected stati.

It has been stated that GEORs can be interpreted at runtime to achieve inter- and intra-application automation that is dynamically adjustable by changing UMEO (→ scalability through flexibility). The question of **who interprets GEORs** has not been discussed yet in much detail. A preceding paragraph used the terms *linking algorithm* and *system*. The reason for this is that there are, in fact, two alternatives: each ProSAp can retrieve and interpret GEOR contents by itself when given situations occur – for example, when the user wants to instantiate a feature class (**decentral approach**). Or a central application performs linking tasks (accessible via services) and is informed by ProSAps about trigger situations relevant for automation. This service is denoted the ***central automation management (central AM)*** here. While both solutions can be adopted in parallel, it seems appropriate to apply only the former for local automation: after ProSAp1 has done its job, ProSAp2 accesses and interprets the results and performs automation controlled by GEORs.

9.4.2 Central Automation Management and Control Information

Bases on what has been introduced in the preceding section, the questions of who performs and controls automation are discussed in the following in more detail.

The second (centralized) solution proposed in the preceding section is, of course, the more powerful one but requires more infrastructure. Here, any ProSAp sends **events** to the central automation management, which in turn decides whether and how to react by retrieving and interpreting GEOR contents. The resulting automation is performed by sending events to the ProSAps, telling them which EO classes to instantiate using which attribute values, followed by instantiations of IEORs to close the gaps between the EO instances. This scenario shall be explained in some more detail below.

Instantiation. This mixed-biased approach requires that a **watchdog** functionality within the ProSAps be triggered when users desire to instantiate a certain EO class into their PPR models. This way, the ProSAp informs the central automation management. The latter reads UMEO and checks it for relevant AutoCreates. Case (a): If there are none, the ProSAp is told to continue instantiating the selected EO class, also using UMEO's information on class attributes, etc. Case (b): If there are ACs, their are interpreted. During this interpretation the applications concerned are told to instantiate certain additional EO classes. More than one application may be involved. Also, relevant EOR instances will be created and stored, as has been mentioned.

Change management. If the user wishes to edit an EO instance, the ProSAp informs the central automation management, which in turn checks the related ProSAp's PPR models by reading the EOR instances and the UMEO contents. Then the desired changes are propagated by the central AM from the initiating ProSAp to the others. The same principles are followed if a user wishes to apply **comments** or other information to EO(R) instances.

As the task of interpreting AC contents may be quite complex, it seems sensible to implement it as an independent application. Alternatively, it may be integrated within the ProSAps or the central AM. Using a **central AM** enables transaction handling as well as collision detection and avoidance. It also opens the way for cross-application automation, i.e., more than one application is requested to instantiate EO classes.

☞ In this solution, AM performs event handling, which is a kind of **control flow** that has been identified as useful for coordinating cooperation, thus making it more effective and efficient.

Suggested solution. As scalability is a central demand from the practical point of view, it is suggested that both alternatives of automation be foreseen and offered: the decentral and the central automation using automation management.

9.4.3 Some Principal Considerations on the Achievable Degree of Automation

During the discussion of the relevant state of the art, also approaches to feature mapping and feature linking have been considered. This example of an automation field is used in the current section to qualitatively show the existence of limits and of appropriate workarounds.

The studies of practical scenarios performed during this research suggest that not all mappings between EO sets can be performed automatically. Some reasons may be (a) difficult modeling due to highly complex interrelationships between EOs, (b) mapping knowledge that is hard to retrieve from the experts, (c) the existence of a wide variety of possible solutions, or (d) a lack of desire to standardize the mapping process caused by a fear of being restricted too much. These cases might be covered better by adding **feature recognition** functionality to the system. For the as yet untreated fractions of the source model, the user might do some kind of interactive **feature identification**: the user may associate predefined feature classes to geometric entities in the product model. Although associating very abstract* feature classes yields less benefit than specific ones do, it at least allows any geometric entity to be treated as a feature (EO). This is crucial for the subsequent step: for EOs generated through feature recognition or manual feature identification, the user has the option to perform **manual feature linking** in order to create the EOR instances necessary for a subsequent flow of information between the applications, as they inter-relate the EO instances created manually to those generated by the system. This hierarchical approach leads to maximum integration of EO-based models along the process chain combined with a maximum degree of automation performed by the software system.

9.5 Providing Building Blocks for the Engineer

This work also addresses the provision of building blocks for engineers.

Based on the methods developed above, flexible means for providing building blocks to engineers are available: offering company-wide **standardized EO classes** such as features is the simplest form. Combining two or more EO classes by EORMs in order to create **EO constellations (EOCs)** designed to be instantiated in one or more product models is a more sophisticated one and allows for providing,

Footnotes

* On a high taxonomical level

in principle, unlimited complex building blocks. Large and complex constellations are also called **templates**. EOC benefits are stated in the section *Automation and Reuse*.

✍ **Critical reflection**. Meeting the ideal **degree of flexibility** in product design is a complex challenge. Engineering templates (and EOCs in general) are a means of reducing unwanted variance; yet, templates prescribing many details of the product (such as the so-called 80-percent templates, as they are supposed to produce results of 80 percent maturity) may also result in too uniform products that are not adaptable to the new challenges of the market (see also *[Dankwort et al, 1997]*). This danger drops with an increasing amount of engineering knowledge integrated within templates. Today, such templates would be called ***intelligent engineering templates***. If it is not possible to cover enough background knowledge within templates, there is a danger of declaring product copies to be product templates.

9.6 New Processes

New IT concepts usually yield changes in a company's processes. Also the solutions suggested in this thesis affect the procedures of product engineering and offer new chances. As these solutions can be applied in numerous ways, it is necessary to find out the best alternatives to use them practically. This section will concisely spotlight some of such issues that are suited to be discussed company- and domain-independently. The selection is far from being complete and is meant to provoke further considerations.

☞ During the demonstration part of this thesis, certain scenarios within which ULEO is already practically applied will be described and investigated.

9.6.1 New-Generation Information Retrieval

This section discusses new chances in information retrieval technology that rely on the new quality of information representation and sharing in a GIS.

Retrieving information from the global information space means finding and accessing desired EOx entities that are potentially correlated to each other and potentially possess different EOx types. Information retrieval is achieved by using GIS services. Navigational information retrieval has already been shown to be, for the most part, adequate here.

On top of this, a more sophisticated sort of information retrieval can be offered – for example, by dedicated retrieval service applications – by utilizing the presence of semantic information to find the best paths within the GIS's information network for finally retrieving the most adequate information. In order to collect information, such retrieval algorithms can follow EORMs or EORIs of types that are a-priori known or identified dynamically. Subsequently, the retrieved information can be rearranged and abstracted to a degree of detail suited for the recipient. Search engines, as known from the World-Wide Web, can be implemented, applying intelligent search algorithms. Such algorithms are also based on sophisticated semantics and context descriptions and feature automatic detection of misspellings, and retrieval and delivery of similar IEs or synonyms (similarity depends on the set of attributes, the set of methods, and the set of relationships (see the section *Context* above).

Through the browsing of instance information, the current state of a company's product development can be determined and reported. Statistics can be provided by doing this on a regular basis.

9.6.2 Dynamic UMEO Contents

This section discusses issues of coping with the dynamic character of the information offered within the global information space and especially of IIM's contents.

Versioning and Archiving

The possibly constantly changing contents of the GIS call for the development of specific strategies for archiving and backing up informational entities.

The GIS's structure is designed to make large quantities of information efficiently manageable in terms of writing, reading, changing, and removing information. The defined basic types of informational entities, however, depend on each other; see materialization and instantiation steps for EOs and EORs from MTRT over UMEO/IIM to the instance databases (ProSAps, EORI). It has been argued that EO(R) instances obtain their semantics from their abstractions in IIM/UMEO and MTRT and that the semantics and context of EO classes and EOR types are comprised of explicit meta-information and of the IE's environment. As a consequence, it is not sensible to store or archive instance information without doing the same with the correlated abstract contents in the IIM and MTRT, which deliver semantics, context, and other meta-information for the EO(R) instances.

A solution to this challenge is the assignment of **versioning numbers** (or other identifiers) correlating the respective IEs. In order to save storage space, the abstract IEs need to be stored only once, as long as they do not change. Regarding what has been stated about semantics and context, the question arises as to which abstract IEs should be stored together with an instance IE and how changes of the abstract IEs are defined (that is, which kind of modifications lead to the "modified" state of a given IE?). This has to be determined for the concrete cases, individually; the most straightforward solution is to regard an instance IE's direct abstract IE only (without its environment of other IEs).

Apart from the issues just discussed, versioning raises some more questions of general relevance:

~ **What is to be versioned at a given time** – all the IIM+MTRT contents or just some specific entities? Answer: As IIM+MTRT potentially change constantly, it seems reasonable to version individual elements. However, if a company's IT environment is rather static in that it changes slowly, it may also make sense to assign version numbers to all IIM+MTRT contents at a time.

~ **What happens to EO(R) instances whose respective abstract information (classes) has changed?** Is it allowed to change them at all? If yes, how can changes be tracked and instances be related to their classes? Answer: Formally speaking, it does not make sense to change classes that are already instantiated and to keep regarding the class/instance relation as still valid. Changes referring to an expansion of a class's list of attributes and/or methods fall into a kind of grey area, as the question of whether it still remains the same class is a philosophical one and has to be decided by considering the practical situation. However, as practical cases may occur where it makes sense to change EO classes such as feature classes also in other manners, it is suggested not to differentiate nor track the conceivable kinds of modification. Instead, classes that have been changed should in principle be regarded as new versions of the initial classes. This means that the changed class has to be assigned a new versioning number and it has to be regarded as not-yet instantiated at the time directly after its modification. The instance information referring to the original class may not be considered as relating to the new version, too. This has to be traced by assigning appropriate versioning numbers to the instances.

~ **Should instances of certain EO classes change automatically when their classes change?*** Answer: This is not in line with the object-oriented philosophy. Thus, another solution should be

Footnotes

[*] This last question was brought up several times by engineers of the automotive OEM mentioned when discussing the usage of user-defined features.

found for this desired functionality. For instance, one product model containing master EO instances that are linked via EOR instances to slave EO instances within other product models could be installed. Changes of the master can be routed down to the slaves utilizing the methodology proposed in the section *Central Automation Management and Control Information*. This solution provides a level of indirection allowing to control the propagation of changes; however, as it also produces much overhead to the data management, it is suggested not to react to EO class changes by changing the EO instances, but to adopt the versioning method described above.

9.6.3 Supplier Integration

Although this work does not aim at developing a fine-grained solution for supplier integration, this issue has been identified to be of vital importance (see the prerequisites for practical use of IT solutions).

Supplier integration is a burning issue in today's automotive development as there are multiple sub-tasks being out-sourced to engineering service companies. Their input, output, and status information has to be transported and optionally transformed between systems. Resulting from the fact that engineering service suppliers typically work for more than one automotive manufacturer, several challenges arise: IT suppliers must be able to cope with various CAx software applications and data security has to be maintained. As the smaller suppliers, in particular, cannot buy all the CAx systems available in the market because of their financial restrictions, information flow solutions have to be developed. Today, standard formats such as STEP are being used for cases in which the automotive OEM's CAx systems are not supported by the supplier.

The ULEO approach provides a foundation for supplier integration. Adopting the ULEO concepts, generally two options are available to exchange (and optionally transform) information between the OEM and supplier:

~ **Online integration**. ULEO does not distinguish the physical location of applications to be integrated into the GIS. In other words, the ideal case of integrating supplier software applications is to consider them standard participants of the GIS. Online integration can be achieved by connecting the OEM's GIS via the Internet or other WAN solutions to the suppliers' GIS, with data security maintained using firewalls, authentication during the logon, and encryption methods such as SSL[*]. The means for integrating ProSAps into a GIS as such are the same on both sides. As this solution provides full GIS integration, individual informational elements and/or ULEO services have to be protected using logon information and access rights for groups, policies, etc. The former is achievable by using related meta-information including free or locked contexts.

~ **Offline integration.** If the suppliers' IT systems are not integrated into the GIS online, integration can be achieved offline by placing supplier integration client (SIC) applications into the OEM's GIS. It is suggested that such a SIC application be specified as follows: the SIC is an application that is capable of generating and maintaining supplier profiles technically describing the informational needs of the individual suppliers, thus partially referencing the IIM/UMEO for a description of the incoming and outgoing information types. The physical information exchange is handled via XML files using the ULEO XML format. Any other representation format can be supported as well. The SIC is adaptable to be applied by any engineer in charge of exchanging information with suppliers, thus providing a view on the relevant information. It should be preferably integrated into the engineer's existing IT tools and enabling selection of the specific set of information that is to be exchanged at a given point in time. Offline integration typically grants

Footnotes

[*] Secure Socket Layer

a higher degree of data security. As extracted files contain copies of the original information, they have to be kept up-to-date according to a given workflow.

9.6.4 Compatible Modeling Depths between Domains

This section tackles a problem of information modeling.

Different ProSAps process information on product, processes and/or resources from different perspectives and on different levels of detail, for example, when regarding the product geometry. While, for example, the CAD system considers the finish-part's geometry down to the level of points and circles, today's PDM systems consider assemblies or subassemblies only – although this might change in the future. Inspection planning applications, however, additionally dive into the very geometric details of the product. This yields the result that not any ProSAp$_1$ is able to deliver information directly referring to any specific type of engineering object (EO class) requested by ProSAp$_2$. Thus, harmonious cooperation and communication between ProSAps must be able to consider this fact in retrieving any other ProSAp's **dimensions of interest** and the according **levels of detail**.

This requirement is met by IIM/UMEO's contents, since any ProSAp's dimensions of interest in the product (in other words, **views on the product**) as well as the level of detail are reflected by its sub-taxonomy of EO classes. If human experts have created qualified[*] EORMs correlating the ProSAps' EO classes such that each of the ProSAps is well informed about the other's informational entities, the above-stated requirements are fulfilled. "Well informed" means that each ProSAp knows[†] which EOR types to follow in order to reach information of a certain dimension and granularity. EOR types in the MTRT describe the kind (and granularity) of information expected on the other side of an EORM or EORI, e.g., by delivering information on the respective OO interfaces.

Footnotes

[*] In the sense of meaningful

[†] By hard-coding or configuration

9.7 Brief Recall of Some Technical Measures

The following sets out a very concise and incomplete list of the more technical measures contributing to the ULEO approach. It briefly recalls these rather specific issues to mind, assuming the reader has read the full text.

GIS information structure:

- MTRT stores EOR types.
- UMEO is part of the IIM.
- IIM/UMEO store EO classes and EOR materializations.
- EOR types from MTRT materialize into UMEO as EORMs. Materialization is a semi-instantiation, where some – but not all – EORM attributes are assigned values.
- EO classes are instantiated by ProSAps into their proprietary PPR models as EO instances.
- EORMs are instantiated inside a global EORI database as EOR instances. All EORI attributes are assigned values.
- IIM and MTRT and EOIs and EORIs form the GIS information contents and are accessible via GIS-wide services.

Communication in the GIS:

- All ProSAps are extended by service-based inter-process communication interface via customization.
- Services employ an optimized, XML-based GIS representation format to transport user information.
- Services use a sophisticated EOx identification and addressing schema to assure robust EOx storage and access.

Automation:

- Automation knowledge is modeled inside EORMs (within IIM/UMEO).
- GEORs (in addition to IEORs) are introduced to facilitate dynamic automation behavior (micro workflows) and thus scalable systems.
- For sophisticated automation, a central automation management is implemented.

Building blocks can be implemented as:

- EO classes
- EO constellations of any complexity; intelligent engineering templates

Chapter 10 Detailing Some ULEO Principles

This chapter discusses a collection of individual questions, not influencing ULEO's major concepts or methods, but coming up when the approach is to be practically applied. Thus, this chapter forms a link between the theoretical development of the approach and its practical implementation.

10.1 Views on Informational Entities

This section discusses the question of how to support views on the contents of the GIS.

For the same informational subset, there can be a discrepancy between the set of IEs that an IT specialist would prefer to use and that an end user would prefer to see: for example, psychological studies performed in parallel by Hartmut Schulze and others (see *[Schulze et al, 1999]*) showed that engineers, while browsing available feature classes for instantiation, tend to literally hop around the IT specialists' ideal world of EO classes. As this user view may also vary from end user to end user, it is concluded that, at least in the case of the engineering objects the user is directly faced with, there should be a method to distinguish between the information-technically ideal IE set and the views of individual (groups of) end users.

☞ Sub-taxonomies and contexts are other kinds of views within the global information space. These are views on the engineering domains' products, processes, resources, etc. (**domain views**). A user's view on the informational entities in the global information space is located on a different level. Hence, views on domain entities (domain views) have to be distinguished from users' views on informational entities describing the domains (**informational entity views**, or **IE views**).

To support IE views, it is not sufficient to simply flag individual IEs within the GIS, as totally different sub-taxonomies might be necessary to describe the way of navigation preferred by a user. The individual elements within these user-specific sub-taxonomies may comprise elements from more than one element in the computers scientist's taxonomy. This suggests that individual sub-taxonomies for each user or group of users need to be maintained and correlated with the GIS's ideal IE sets (domain view = system view): the latter should be arranged according to the IT specialists' preferences, as will be motivated in the following.

Such typical view-based approaches can be **implemented** in various ways: The system view could be one of many peers or the central and most important one playing a kind of master role. The former is an interesting philosophical approach as it does not call for a master view at all and points out that all information modeling is subjective; yet it complicates information management too strongly and is thus costly compared to the practically achievable benefits. A view on a given set of inter-related informational entities might be a subset of this set or a newly formed correlation of its elements (where the view's creator has to consider the validity). Therefore, the central information content is not brought to discussion; instead, the presentation is to be adapted to certain needs. Taking this into account, a central system information pool seems to be valid. Using modality meta-information, varying information contents can be realized more naturally. Based on a vote for a system master solution, …

~ the individual views could allow fully independent IE sets or just flag out certain elements within the original master sets. The latter has already been argued to be insufficient.

~ the views on the master IE sets could be stored locally and externally from the global information space – i.e., within the responsibility of the ProSAps' proprietary information handling. This keeps the system IE sets lean but hampers the management of views that are of common interest (group-based views) and raises the question of how to relate information sets outside the global information space (views) to such inside it ("original" information). The alternative is that the view-specific subsets be regular parts of the global information space marked by corresponding context information and related to the "system" IE sets by relations. For many users, many views might be desirable. If views on UMEO are stored within UMEO, it could potentially be blown up significantly. Views on ProSAp-managed instance information can hardly be managed by the respective applications themselves, as the number of information users might be large. Although each of the options discussed raises questions or problems, the latter shall be promoted.

Further discussion. As an individual view on a set of information is not meant to represent different information contents, it is a description of the presentation of a given set of information. In this sense, it is meta-information added to what has been called system information above. Such **view meta-information** (**VMI**) can refer to abstract or to instance information. As such information is managed according to ULEO in different locations and by different systems, a method that copes with view meta-information on distributed system information has to be found. In addition, the users of this meta-information might be any ProSAp or service application[*]; it is therefore ProSAp-generated and -specific information by nature. This includes groups of ProSAps sharing the same VMI. Another issue is that such view meta-information might be intended for just one user, but also for groups of users. Finally, view meta-information might have to be adjusted to changes of the system information. This indicates the need to discuss the usage of **rule-based** information for dynamically creating desired views, thus avoiding recreating the views after each change in the system information. This method, however, cannot generally be adopted, as not every view might be representable by means of rule-based expressions. Hence, views promise to be efficiently usable for stable information only. Such stable information might be found within such areas within the IIM that have been subject to pertinent standardization agreements (see *semantic kernels*, above).

Suggested solution. Taking into account the issues raised in the previous paragraphs, the following approach is suggested for implementing user-specific views on any kind of “system” information within the global information space:

- From the viewpoint of a ProSAp that has to offer specific views to its users, the view meta-information (VMI) replaces the system information in that it is browsed instead.
- The VMI is represented using the same basic information types (EOx) as employed for the system information.
- The VMI is a set of IEs optionally containing references to the system information. Individual VMI entities might also represent additional navigation-structuring information and not refer directly to system information.
- References from VMI to system information can be rule-based expressions within the VMI elements’ attributes or methods or can be dedicated types of (a) EORMs for abstract information-related VMI or (b) EORIs for instance-related VMI, respectively.
- As VMI can generally be of interest for more than one ProSAp (independent of the question whether it is class- or instance-related), it should be stored in a globally accessible location. In cases where VMI is purely single ProSAp-specific, it can also be handled locally by this ProSAp. For the general case, it is suggested that the class-based VMI be stored within UMEO and instance-based VMI be stored within the EO(R) instance databases[†]. The applications using VMI are responsible for its creation and maintenance[‡]. The potential growth of UMEO can be accommodated by efficient hardware and software. In order to achieve satisfactory response times for GUIs, strategies for retrieving the VMI can be applied by the applications utilizing it.
- Where should VMI be represented inside UMEO? All VMI could be stored in a central sub-taxonomy, or it could be stored distributed all over UMEO and closed to the related system IEs. In principle, both alternatives are valid. The former, however, allows a better structuring of the

Footnotes

[*] See the implementation part of this thesis

[†] See the implementation part of this thesis.

[‡] Note that these are not necessarily the applications whose information is attached with views.

knowledge base as it encapsulates the view information more than the latter does. Therefore, choice of the first alternative, i.e., storing all VMI within a central sub-taxonomy in UMEO, is suggested.

☞ Views are considered an optional means for filtering and re-arranging information and are assumed not to be generally necessary for each ProSAp. On the contrary, it is expected that only few areas of the IIM, in which users directly browse, will have to be equipped with views.

10.2 Avoiding GEOR Cycles within UMEO

This section covers another issue of information modeling.

As generative EORs automate the use of other IEs within UMEO, cycles may arise from a chaining or cascading of several GEORs and lead to an infinite automation loop. Such effects can be detected most easily, if all coherent automation steps are handled centrally, i.e., by a single application. In this case, the automation service can log trigger conditions that have been processed within a certain automation sequence and stop processing on its n-th encounter. To avoid such late findings, it is advisable to implement algorithms that will detect such problems in advance.

However, there is a general problem with detecting cycles, as cycles may also be intended. Thus, the detecting algorithm should ideally be able to understand the intention behind it. Although it is not likely to generally achieve this, GEORs could carry respective meta-information supporting the detection of wrong tracks.

✍ For practical deployment in automotive product development, frequent problems of this kind seem to be rather unlikely, as responsibilities for all steps within the product development processes are held by engineers who have to understand what happens automatically. This fact suggests that it is unwise to implement long automation chains. Instead they should be kept short and manageable. If automation should become more complicated after practical experiences have been made, the authors of the GEORs and the interpreting software will have to address these questions. Giving a general answer to them is not in the scope of this work, as ULEO will be widely applicable in practice without it and any explanation would exceed the resources available for this work.

10.3 Instantiation of GEORs

This brief section addresses whether generative EORs should be instantiated.

GEORs do not actually represent domain knowledge but describe how to use it. Thus, it is not suggested that instances of them should generally be created. However, in cases where it is important to trace automated steps in product development, the instantiation of GEORs is one contribution to achieving this: a GEORI can correlate all EOIs and IEORIs that have served as trigger information sources, or that have been instantiated automatically during interpretation of a specific GEOR.

10.4 Multiple Inheritance and OO Interfaces within the IIM

The way multiple inheritance and interfaces are adopted leaves room for the information modelers. This section gives some background information.

In cases of multiple inheritance, one child class is derived from more than one parent class. Multiple inheritance is a part of the object-oriented paradigm and allows a further reduction of redundancy in taxonomies in addition to the effects of single inheritance. The properties of a class can be abstracted following multiple perspectives reflected by parent classes. Thus, different groups of classification criteria are separated, and modeling becomes more compact. One practical consequence of allowing multiple inheritance is that knowledge of several areas – reflected in dedicated sub-taxonomies – can be utilized by creating object classes derived from the parent classes of both sub-taxonomies.

- ✰ One practical example where multiple inheritance shows benefits are features that are used within more than one domain and that obtain (i.e., inherit) specific properties from these domains' classes.

However, multiple inheritance also has a drawback in that it can make reasoning **non-monotonic**, i.e., influence information processing and retrieval. Thus, the decision to utilize multiple inheritance in the IIM has to be harmonized with the practical needs of a company.

Can object-oriented OO interfaces substitute multiple inheritance? The IIM's EO classes – as EOR types in MTRT – can implement **OO interfaces**. An interface in the OO sense collects a set of attributes and/or methods and resembles – but does not represent – an EO class. The difference, however, is that while an EO class stands for domain objects, OO interfaces do not necessarily have a counterpart within the domain. They are usually only abstractions on the set of EO classes and are intended to help make the entire model uniform by suggesting same attributes and methods for the same aspects of EO classes such as their visualization in certain environments or means of making them persistent. They are especially relevant for the dynamic processing of EORs (relation-based navigation). Hence, an OO interface generally can not substitute multiple inheritance.

10.5 Variants of Modeling EO classes

The GIS information structure describes EOx classes as objects embracing attributes and methods, with attributes and methods being intrinsic parts of them. An alternative approach toward solving certain modeling challenges is discussed in the following. This approach is applicable while keeping and making use of the standard approach.

An alternative to the above-described strictly encapsulated approach of modeling EO classes is to describe EOx properties as IEs of their own. While the original solution makes information flow more efficient as less IEs have to be exchanged, the alternative solution makes individual attributes accessible for relations from outside. This is interesting when information outside a class refers to a subset of the class's attributes only.

- ✰ For example, a tolerance might refer to part of a feature's geometry, for example, only to the uppermost part of a stepped hole in a cylinder head.

Part V – DEMONSTRATION: IMPLEMENTING AND APPLYING ULEO

As this research has been performed in a hybrid scientific-industrial environment, it has been a major concern to satisfy scientific demands on the one hand and to strive for practical usability of findings on the other. In this part of the thesis a software architecture will be proposed reflecting the insights gained during the conceptual work. Thereafter, its realization and application will be reported.

”The proof of the pudding is in the eating.”

(Ancient proverb – it has been traced back to 1300 and was popularized by Cervantes in his *Don Quixote* of 1605 – often used by Prof. dr. ir. F.J.A.M. van Houten.)

Chapter 11 The Proposed Software Architecture

The ULEO approach can be practically implemented in various ways, depending on the functionality desired and the practical restrictions. Certain philosophical preferences of the IT personnel responsible may also play a role. Therefore, the architecture described in this chapter is one amongst several alternative solutions; nevertheless, it is theoretically ideal in a sense that it is suitable to support the approach's full functionality and reflects also the practical demands such as scalability and flexibility. This architecture has been practically implemented within an automotive manufacturer's product development process chain.

11.1 Overall Architecture

This section derives and motivates the key characteristics of the proposed ULEO architecture.

During the discussions in chapter Part IV – Chapter 9 the following GIS services have been proposed:

- MEO services, including the support of external referencing
- MTRT services
- EORI management
- EOx address resolving
- Central automation management (central AM) including the cooperation management
- Management of the table of semantic description languages (TOSDL)

Bundling this central and GIS-wide functionality within a **single application** reduces the overhead for communication and synchronization between the sub-services and raises the overall efficiency of the system. Moreover, relationships exist between the services such as the UMEO address resolver on the one hand and the external referencing mechanism on the other.

Thus, a **centralized architecture** (see Figure 26) is adopted, whose central element is an application offering all above-mentioned GIS-wide services. This application is called **ULEO server**. Nevertheless, depending on the total number of ProSAps involved and on the physical network infrastructure, it may be beneficial to use more than one ULEO server in order to distribute each single's work load and network traffic. This method promotes scalability. The same holds true for the **databases** involved: since their physical location and realization are abstracted by the ULEO server, legacy databases may be used. However, in the general case, they do not have to be available and UMEO, MTRT and EORI contents are stored within a dedicated *ULEO database* (Figure 27 shows first MTRT contents). Also the ULEO database may be distributed as suggested for the ULEO server. However, for the first practical introduction of a ULEO system it is suggested to start off with an non-distributed and in this sense integrated and centralized architecture foreseeing a single ULEO server accessing a single ULEO database – this allows for a higher transparency and easier traceability of the system behavior.

Also the **online communication** between ProSAps can either be centralized utilizing the ULEO server or performed de-centralized, directly between every two ProSAps. Although the second alternative shall not be declined for every application case, it is proposed that the communication is generally performed exclusively **over the ULEO server**. This is motivated by the desire to keep ProSAps' software implementation comparatively straight-forward (the ULEO server is the only communication partner), stable, and independent form the set of ProSAps available at a time. Thus, the entire GIS becomes more independent from specific ProSAps and the information as such will be put more into the focus of attention; ProSAps in need of information do not really want to care about which application delivers the information.

✍ According to this method, desired information is requested by calling a service offered by the ULEO server and by merely passing the respective EOx address (e.g., the domain ID *detail* *design*). Consecutively, the ULEO server decides which specific other application to request for delivering this information.

To maintain this methodology any ProSAp may in principle use and offer (almost) any GIS service defined in the ***ULEO inter-process communication interface (ULEO IPCI)***, after is has informed the ULEO server during its own start-up by calling the *logon* service (see Part VII – Chapter 25 in the appendix for a detailed list of the most relevant services).

✍ The terms “ULEO services”, “ULEO IPCI services”, “IPCI services”, and “GIS services” are synonymous.

☞ The ULEO server also acts as a client calling services offered by ProSAps.

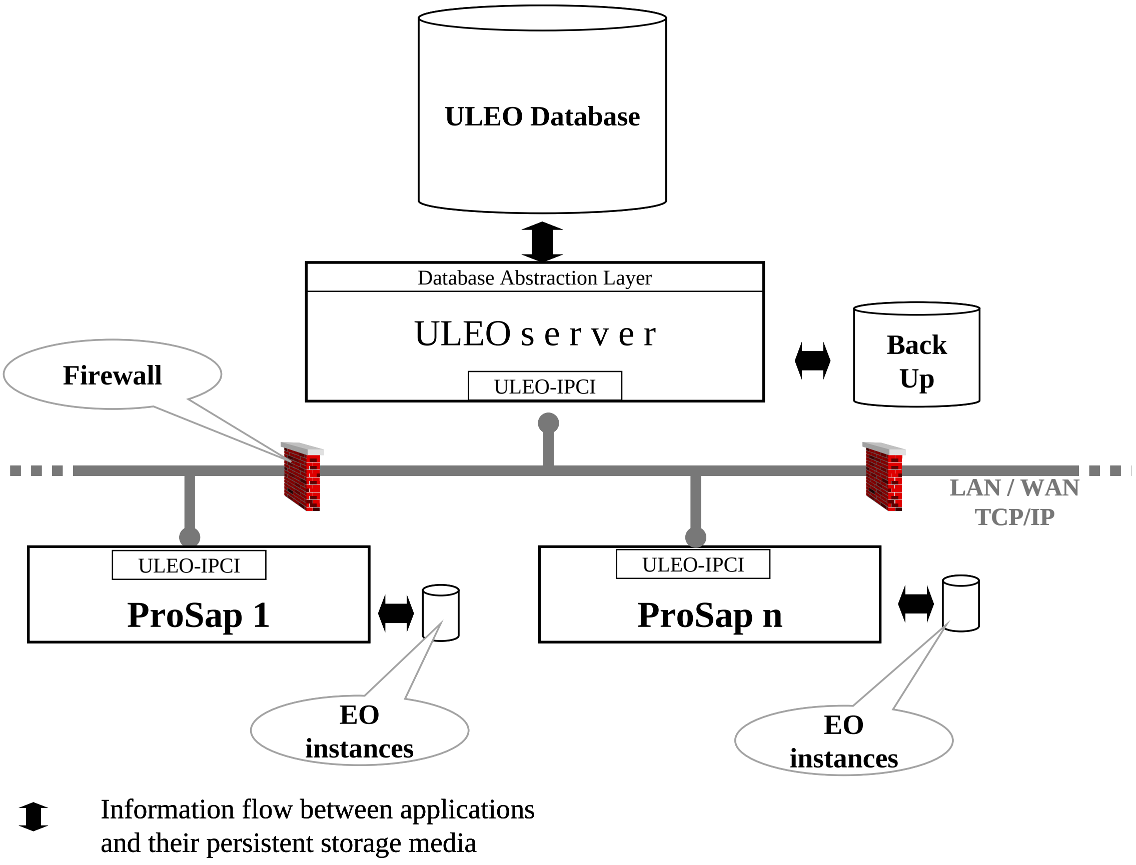


Figure 26: The Proposed Centralized ULEO Architecture

In order to keep the set of GIS services offered by the ULEO server extendible without changing its source code, the server can utilize independent **service applications (SAs)**. They communicate with the ULEO server utilizing the same IPC interface. Their functionality might be used internally by the ULEO server or made available to the ProSAps (via the ULEO server) by one of two methods: (a) the IPCI is extended by additional GIS services (requires software changes in the ULEO server) or (b) a universal *BaseCom* GIS service is utilized. Changing the IPCI, case (a), is proposed if complicated and detailed additional services shall be introduced that are expected to be used frequently by ProSAps.

☞ Offering a universal service suggests implementing a service resolution and routing functionality (**service mapping**) within the ULEO server.

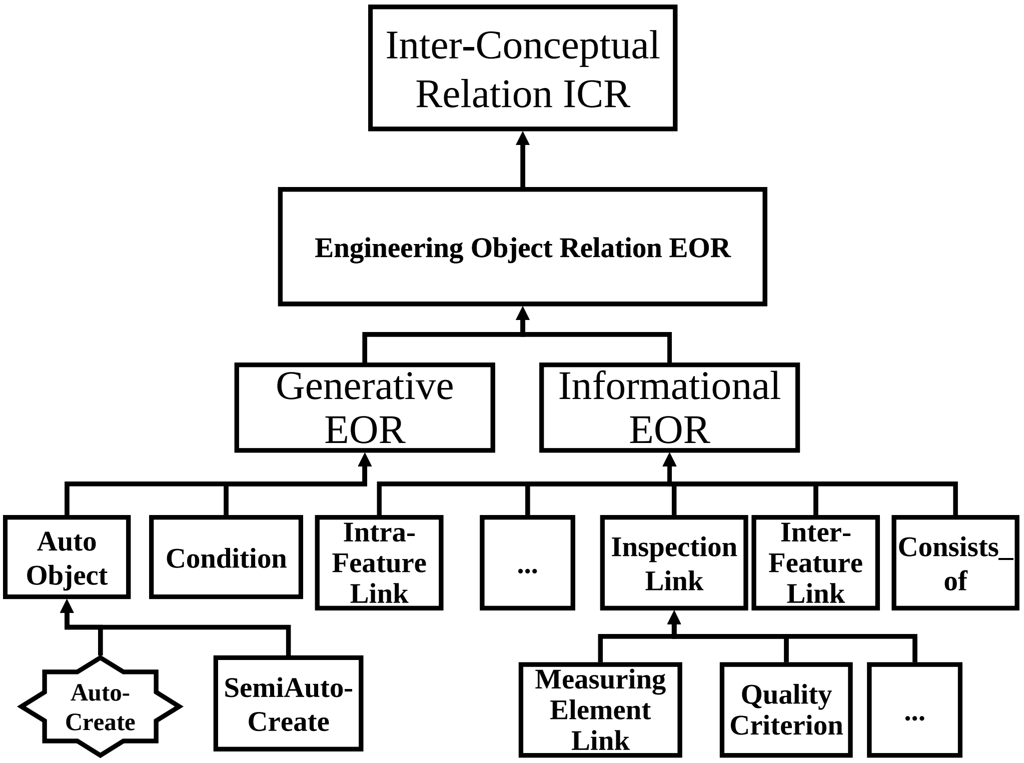


Figure 27: First MTRT Contents

- ✩ **Service applications**. In practice, ProSAps can be supported by powerful services such as cost estimation, workflow management, and management of experiences and justifications. Other examples for ULEO service applications are components offering user modeling and knowledge processing (i.e., reasoning) functionality or constraint solving capabilities.

A dedicated **ULEO XML format** (see section *Information Representation Formalism* on page 127) has been introduced for representing information within the global information space. Since each informational entity may carry any set of attributes and methods, highly varying information has to be storable within the above-specified databases. To avoid conversion of information between the databases' internal schemas and the ULEO XML format, it is suggested to store it straight-forward as text blocks in the ULEO XML format.

11.2 Details

The current section elaborates several aspects of the chosen architecture to the degree of detail necessary to obtain a realistic picture.

11.2.1 GIS Representation Format ULEO XML

This section provides a brief overview of ULEO XML, which serves as the central representation format of the GIS. The development of a pre-release of the ULEO XML schema (UXS) is described in [Van den Elst, 2002]. P*lease refer to the appendix Part VII – Chapter 24 for some further extracts from the ULEO XML schema.*

ULEO XML is applied (1) to represent user information within the GIS service calls, (2) to store information in ULEO databases and (3) within offline XML files.

☞ It has been argued above that there is no state-of-the-art representation formalism suitable to meet the requirements stated in this work. Although it would have been possible to ignore this fact and use some existing formalism, this work strives for an optimized solution.

In order to achieve sufficient runtime behavior, the resulting solution has to be as general as necessary for the given purposes in the GIS and as optimized as possible. This implies utilization of XML-specific functionality such as XML data types. It also implies representation elements reflecting the GIS structure. Thus, ULEO XML is optimized (1) for maximum semantic content, (2) for efficient processability and (3) for meeting other requirements stated above, while – on the other hand – its expressiveness maintains the variance of representable information, as specified above. Figure 28 shows the typical structure of ULEO XML blocks.

✰ To further illustrate this principle: ULEO XML offers dedicated XML data types for representing the EOx identification and addressing schema, and IEs' meta-information including contexts and explicit semantics. Additionally, ULEO XML provides enumerated types for documenting an IE's basic type such as *EOinstance* or *EORM*. Figure 29 shows another example, the *LogOnInformation* data type, which is used for the ProSAps' logon at the ULEO server.

☞ XML files adhering to the ULEO XML schema may in principle carry the GIS's complete information content or any part of it. The same holds true for UXS-formatted XML strings that are used within the GIS services.

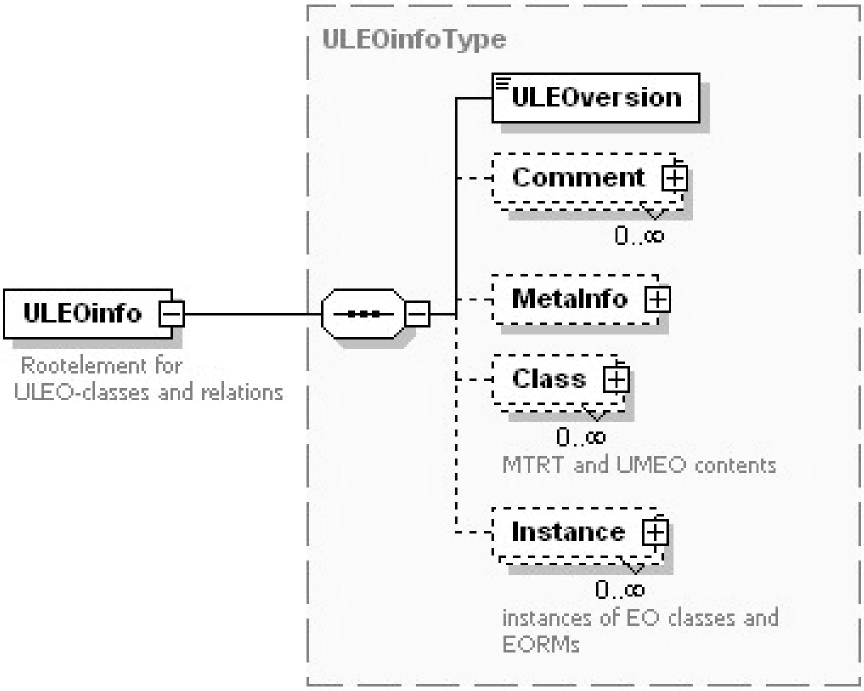


Figure 28: Uppermost Node in Most ULEO XML Blocks

Outline: Global structure of ULEO XML files. The root element of UXS-formatted XML files is called *ULEOinfo*. Any ULEO XML file contains exactly one such element. Within the *ULEOinfo* element, further dedicated elements support the specification of the UXS version and the representation of any kind of comments and of meta-information. Subsequently, any number of uniform *Class* elements may follow. Each of them may describe exactly one EO class, MTRT class or EORM. A sequence of them is suited to represent complete or partial contents of UMEO, MTRT or any other MEO's contents in the IIM.

Behind the *Class* elements, any number of *Instance* elements may follow each of which carrying information on a single EO instance or EOR instance. For example, a list of EOR instances is suited to represent complete or partial contents of the EORI database.

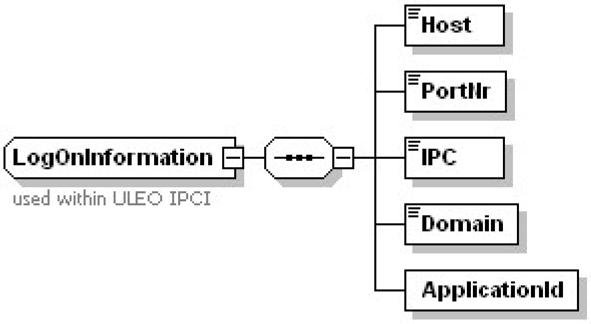


Figure 29: XML Data Type LogOnInformation *within the ULEO XML Schema*

11.2.2 ULEO Inter-Process Communication Interface

In principle, services can be offered and used by any application involved in the global information space. This section investigates how this can be technically achieved and which services should be offered in detail.

CORBA and **WebServices** have been identified as the most common means of inter-process communication. As the basic functionality of CORBA and WebServices are comparable to each other, and as automotive companies tend to have fix regulations concerning IT deployment, it is suggested to offer both variants within the ULEO server, while ProSAps can decide which solution to chose.

✍ Synchronous http-based communication such as the WebServices over http suggested here is subject to **time-out limits**. This kind of communication is thus not suited for long service calls.

Distributing Functionality between ULEO Server and Service Applications

In principle, any application within the global information space may offer and use services (Figure 30 shows a typical situation). The ULEO server plays a central role by mapping service calls of ProSAps on services offered by other ProSAps or by service applications. The question arises, which set of services should be built into the ULEO server, and which services are better done by dedicated service applications. As with several other design decisions, it is not possible to propose the optimal universal solution here; however, it is possible to reveal the relevant decision criteria and to propose a standard solution for the most probable use cases.

Important decision criteria are, amongst others, the **frequency** by which certain functionality is needed, thus influencing the overall system’s efficiency and the **degree of specialization**: highly specialized services, e.g., used by just one ProSAp, suggest themselves not to be included into the ULEO server, which is a universally applicable software application. Only common and frequently needed functionality should be included into the ULEO server application.

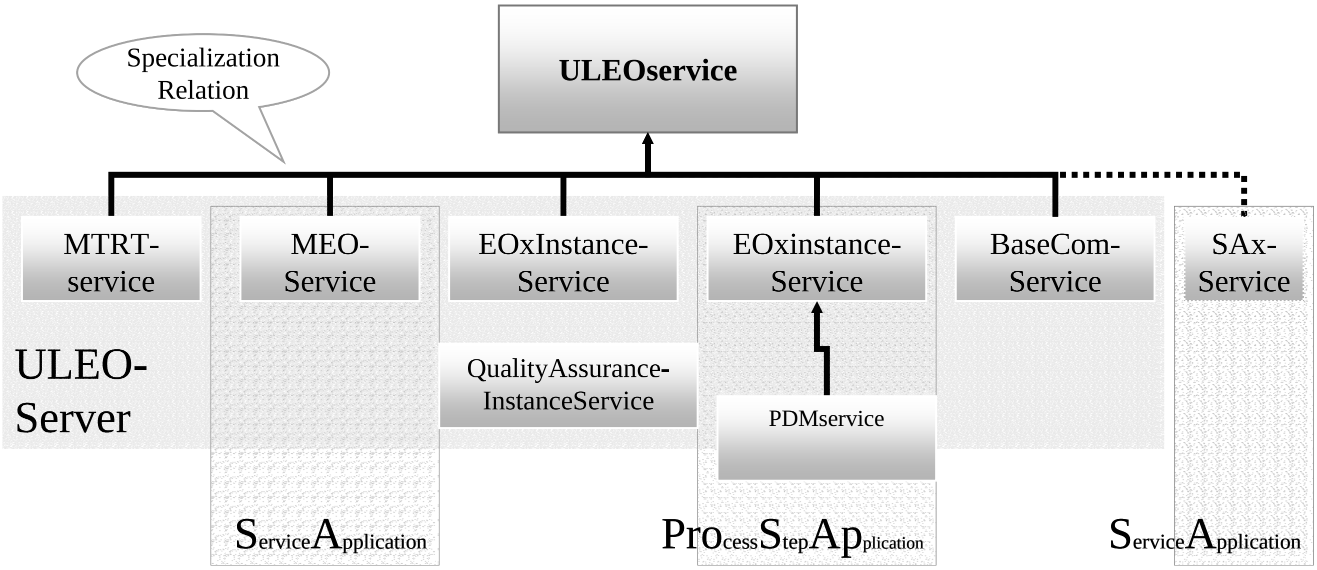


Figure 30: GIS Services and Offering Applications

Sticking to this line and considering what has been discussed in the preceding sections, a **standard** set of **ULEO server functionalities** is suggested as follows:

- Collect EO attributes and methods from parent classes and interfaces within the taxonomy by following inheritance relations
- Collect EORMs for a given EO class or EORM, including the support of EORM chaining via EOx paths
- Determine attribute values by following inheritance relations and considering defaults, fixed values and overloading
- Resolve external references
- Manage EO instances for applications without own data storage
- Manage EOR instances
- Handle events (control flow) and perform basic GEOR interpretation

Event handling is relevant to support automation functionality. The ULEO server should be able to receive events from ProSAps, to locate relevant GEORs such as AutoCreates within UMEO, to interpret their trigger conditions and, if needed, to set off corresponding actions by executing service calls to SAs or ProSAps (e.g., on EO instantiation).

Performing **more sophisticated types of reasoning** (knowledge processing) such as the resolution of implications of inheritance relations does not seem to be common enough to be included as a base functionality into the ULEO server – although this depends on the company's specific situation. To have reasoning performed, the ULEO server provides a dedicated service application with the relevant IIM/UMEO contents and retrieves the desired information that has been produced by the SA's knowledge processing. Depending on the frequency of retrievals of specific IIM portions, the SA may chose to retrieve these entities either once on startup or before each individual reasoning task. The IEs' context information supports the identification of such IIM information that is relevant for the SA to be able to answer a specific query. Further meta-information can, for instance, help to specify the degree of reliability of information and to filter out incomplete[*] information.

Footnotes

[*] In the sense of unfinished

Proposed Set of GIS Services

The following basic set of services is proposed to offer the functionality needed to allow applications to co-exist in a global information space and to intensively make use of it:

- **MEO services** (model of engineering objects) for accessing and manipulating contents of UMEO and other models in the IIM
- **MTRT services** for accessing and manipulating contents of the MTRT
- **EOx instance services** for accessing and manipulating EO(R) instances
- **File services** for backward compatibility to legacy ProSAps (where file-based information transfer is usual); references to files can also be useful within UMEO.
- **BaseCom services** to ensure extendibility of the set of services
- **ULEObase services** for allowing ProSAps to register with the ULEO server and retrieving available services; they also offer transaction handling (start transaction, end transaction, rollback).
- **Administration services** to support management of users and groups and their access rights to services, for triggering of logging and statistics as well as for performing archiving, and versioning; They also comprise imports and exports to and from IIM/UMEO and other GIS information from and to ULEO XML files.
- **Test services** for providing test functionality for new ProSAps such as *echo*.

On top of this standard set of GIS services, it is possible to introduce additional services to provide more comfortable functionality to ProSAps of specific domains such as quality assurance or for accessing product data management information.

✍ The proposed arrangement of services into groups is not the only feasible solution but the result of an ordering according to informational base types or according to similarity of administrative functions, respectively.

In Part VII – Chapter 25 in the appendix, the most relevant GIS services will be listed in more detail.

11.2.3 ULEO Server

This section discusses some internals of the ULEO server and its cooperation with process step applications and service applications.

The ULEO server must support **multi-client** operation, as at a given time, several ProSAps and/or users[*] may call services to be handled by the server.

Although the key functionality of the ULEO server (reflected in the internal structure sketched in Figure 31) happens hidden from the users and ProSAps, there has to be an **administration component** including a user interface for purposes of common administrative tasks such as direct manipulation of database contents, consistency and redundancy checks on the database contents, backup and versioning, management of users, licenses, and access rights. Also the viewing of logging results regarding system failures, access logging and statistics are amongst its functions.

Footnotes

[*] Each ProSAp can provide several user-specific sessions, each of which can individually call and offer GIS services.

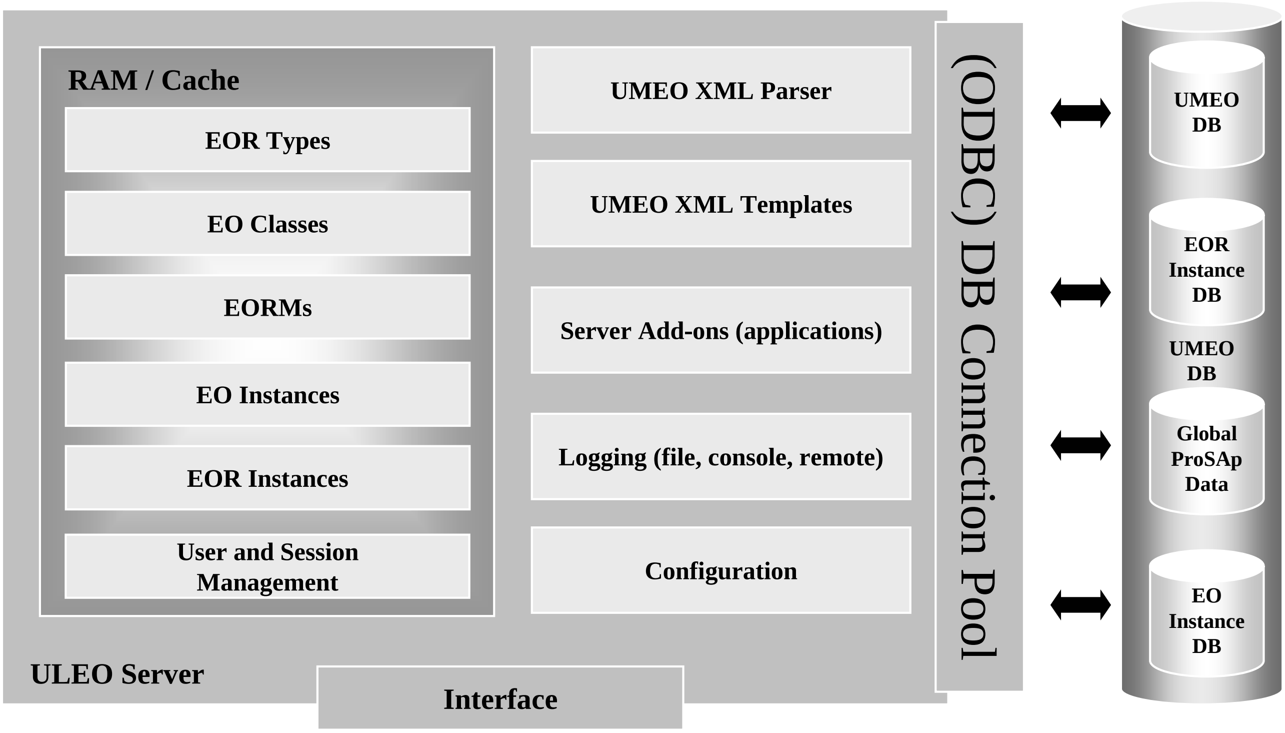


Figure 31: ULEO Server – Main Components [J. Mellens]

Application Logon at the ULEO server

In order to make services generally accessible within the global information space, all applications offering services inform the ULEO server about this fact (usually) on startup by logging on at the ULEO server. This registration helps also to raise the level of security within the global information space, as an application may not exchange information with others until having been affiliated by the logon procedure. The centralized architecture supports this concern, as all GIS communication happens via the ULEO server.

When calling the *LogOn* method of the ULEObase service group, each application specifies its unique application ID[*], the desired IPC type and TCP/IP port as well as the domain it is assigned to, and obtains a unique **logon ID** from the server (see Figure 29). The same application may be run several times on the same host if using different ports. The logon ID is used as an input parameter of the *GetLogin* method returning a unique **session ID** identifying the current user of the ProSAp to the ULEO server. The session ID is used within almost any service call to uniquely identify application and user. One single application can log on several times using different domain identifiers. Within a given logon ID, more than one user can work, which may lead to several session IDs per logon ID.

Footnotes

[*] This is the application name described in the sections *Identification and Addressing Schema* and *Context* above.

- ☆ If, for example, a certain application hosts more than one dedicated sub-applications such as the so-called *workbenches* of the CAx system *CATIA*^{TM} *V5*, each sub-application can get its own logon ID. As these workbenches can run in parallel, the user is able to perform tasks of several process steps in parallel. For the global information space this situation equals to that of several independent applications.

The ULEO server internally stores the log-on information within the ***Logged-on Applications table (LAT***, see Table 2) and the session information within the ***Session table*** (see Table 3).

Logon ID	Domain	ApplID	IPCtype	host	port	Services
01122	Background-Knowledge	OW	WEBSERVICE	abc	xy	MEO, EOinstance
02331	Background-Knowledge	Expert-System1	CORBA	888	999	MEO
01231	Unique-Domain1	-----	WEBSERVICE	878	665	MEO

Table 2: Logged-on Applications Table (LAT)

Logon ID	Session	UserID
01122	0112s	ben

Table 3: Session Table

Extending the proposed modus operandi, also **domains** can be managed in a **hierarchy** that allows for a efficient representation of domains within the address fields but slows down their processing at runtime: the database's join functionality is not usable, as it does not know the hierarchical dependencies between domains.

- ☆ For example, the domain *inspection* could subsume the domains *inspection planning*, *inspection programming*, and *inspection analysis*.

In the inspection scenario described here, domains are not treated hierarchically.

EOx Address Resolution

Address resolution assures a robust access to informational entities. As discussed in the sections *Identification and Addressing Schema* (on page 122) and *Context* (on page 120) above, the global uniqueness of IE identifiers will is not required and would hardly be feasible, as each ProSAp manages its instance data by itself and thus assigns own identifiers to them. Managing contexts for informational entities has been identified to solve this and other problems. To shortly recall the principle, each informational entity in the global information space is uniquely identified and addressed by an ***EOxAddress***. An EOxAddress is a filled identification and addressing schema built into ULEO XML (see Figure 22) and consisting of a locally unique *identifier* plus *context* information, where "locally unique" means unique in the scope that is specified by the context information. The context structure, in turn, consists of a *domain* identifier, a *locator* specification, and an optional *application* name.

✰ Some examples are given here:
Domain = *DetailDesign*, *Inspection Planning*, *UMEO*, *BurrMinimizationKnowhow* etc.
Locator = \\...\..\x.catpart→part.001→mypart
Locator = Smaragd→Baureihe X→Version y→part z→hole1
Locator = UMEO→EO.Feature.DesignFeature.Holes.ThroughholeType2
Application=CATIAV5 – if CATIA^{TM} is not the only system within the specified domain

These examples illustrate that a **locator** string may consist of several portions, delimited by a certain substring. The use of XML for structuring the locator is optional, but for the implementation within the inspection scenario (see the section 13.3.1) it has been regarded unnecessary, as substructures are simple and processing takes time. The addressing schema is mirrored by the IPCI. *EOxId* is the locally unique identifier of the IE; *Contexts* is a list of context specifications identifying the location of the respective IE optionally through multiple retrieval steps. Each context consists of *Domain* and *Locator* and a universal substructure for future use (see Figure 23). While processing a service call, the ULEO server interprets the address context structure to stepwise retrieve the destination.

The ULEO server is able to determine and locate the application, able to deliver or accept a given informational element by comparing the list of services, the domain, and optionally the application ID from within the address context to the *logged-on applications table*, from where it gets host and port information. If more than one application serves a given domain, the ULEO server entrusts them one after the other in order to distribute the load. For certain company-specific situations, other solutions may have to be taken. It is also conceivable to define the domain names in a way that avoids multiple applications offering the same services at a time.

✰ The application ID is used within the inspection scenario below to identify a certain EORM to be specific for a certain ProSAp called *MPE*. In this example this EORM is used just for supporting the user interface behavior but is not of common interest within the global information space. Nevertheless, this EORM is embedded in the global information model. For this purpose, all EOx in UMEO belonging to the I++ information sub-model are designated by one or more domain identifier from the following list: *finishpart_design*, *assembly_design*, *inspection*, *inspection_planning*, *inspection_programming*, *inspection*, and *inspection_analysis*. The above-mentioned EORM's address struct, however, carries merely one domain identifier *inspection_planning* and additionally the application ID *mpe*. Thus, the general I++ information model is kept generally valid, while the specific ProSAp gets the helpful relations anyway.

11.2.4 Databases

Tables and the schema of the ULEO database are briefly presented in the following.

Although there are database systems specialized for managing XML data, the use of a **relational** database for storing XML-based GIS information is preferable, since this universal and flexible **solution** avoids the need to adapt the ULEO database to each specific object type; the special benefits of XML databases cannot be deployed here.

In order to be even more flexible in the choice of the implemented database system, **ODBC** or a similar quasi-standard for accessing databases should be chosen for communication between the ULEO server and ULEO database.

The **structure** of the **tables** inside the relational ULEO database can be kept quite simple and therefore general enough to permit future changes. Each EOx can be stored as an XML text entity within a single table cell. Each database record can carry copies of the most relevant key information in dedicated additional fields, thus enabling quick access to the data.

Figure 32 depicts part of the entity relationship diagram for the ULEO database's **major tables**:

- *MTRTClass* + *MTRTrelation*[*] describe **MTRT contents** (taxonomy of EOR types).
- The **UMEO contents** are stored within the *EOClass* and *EORM* tables. The EORM partners are stored in separate tables for faster access: *EORMEOpartner*[*] + *EORMEORMpartner*[*].
- **EO instances** are stored in the *EOinstance* table, **EORIs** are stored in the *EORinstance* + *EORinstancerole*[*] tables.

While the ULEO server prototype operates with a Microsoft Access™ database, the productive version utilizes IBM's DB2 database running on a UNIX system.

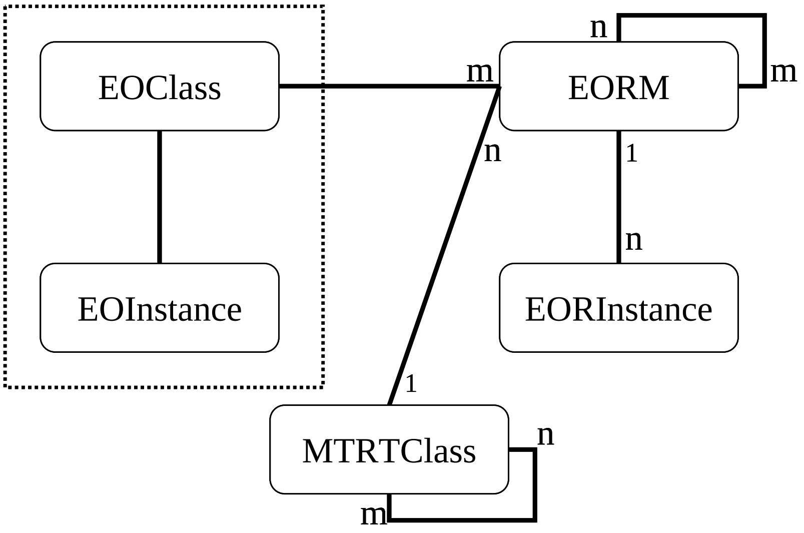


Figure 32: Simplified ER Schema of the ULEO Database [J. Mellens]

Footnotes

[*] Not depicted in the figure

Chapter 12 Variants of the Proposed Architecture

Meeting the requirement for scalable IT solutions, ULEO may be applied in several variants that differ not only in terms of the complexity of the software systems' internal processes but also in terms of their capabilities regarding information flow, automation, and user support. The respective architectural characteristics will be pointed out below.

☞ *Chapter 11* showed that all the requirements formulated in the *catalog of requirements for IT solutions for engineering* can be met within a single approach in a balanced way (denoted as "full" ULEO below).

Not restricting this finding, the two following limited variants were introduced because of their immediate usability in product engineering together with the low effort required for their implementation.

Manual EO constellations (MEOCs) are a comparatively simple approach investigated, as they are implementable in the context of a state-of-the-art CAD system – MEOCs can, for instance, be implemented using the *User Feature* or *PowerCopy* functionalities of the CAx system CATIA^{TM} Version 5 (see *[Ourari, 2001]* and section 13.2.1 for the description of the respective prototype). MEOCs are geared for a **partial automation of EO instantiation** accelerating the engineers' work and promoting the avoidance of errors*. The main notion is to predefine multiple EO classes and their parametric dependencies as kinds of patterns stored within dedicated libraries (corresponding to and replacing UMEO; there is no correspondence to MTRT and no ULEO server). The user can instantiate such patterns into a product model called the *base model*, which is dedicated to this task. From here, the individual elements such as features or parts can be *copied with link*† to their final destination models. Changes to single components have to be done within the base model (see also the section *Multi-Modeling Technique and Vertical Associativity between CAD Models* on page 20).

EO constellation linking (EOCL) is not implementable using current CAx systems without adding code; it is, however, still a local solution requiring a rather small implementation effort. In contrast to the MEOC method, no base model is needed and users do not have to manually copy instances into their final destination models, since the system will do this for them‡. The CAx system modified by a software-add-on performing EOCL reads EOR-based information from UMEO to find out corresponding components of EO constellations – however, the meta-information on the supported EO constellation types is hard-coded into the software-add-ons. Subsequently, the add-on automatically performs the instantiation into the destination models, and generates and saves also those EOR instances representing information on the EOIs' affiliation to EO constellations. Therefore, in comparison to MEOC, EOCL enhances capabilities for the integration of PPR models and for automation. The correspondence to UMEO and the EORI database can be managed using any UXS- or table-based storage (see *[Ananthanarayanan & Addala, 2002]* and section 13.2.2 for the description of the respective prototype). There is no correspondence to the MTRT, and still no ULEO server.

Footnotes

* Slips of the pen

† Uni-directional associations offered by the CAD system.

‡ Instantiation of multiple EO classes may occur into one or more product models.

✍ The unrestricted ULEO architecture, as proposed in Part IV – Chapter 9, adds still significantly more functionality to EOCL. Only full ULEO supports the GIS philosophy and services. As to automation, this means, for instance, that full ULEO is not restricted to the processing of specific predefined relation types, and that it adds the representation and access of engineering strategies and the handling of GEORs and events. ***Linking on the fly*** facilitates automated generation of EO instances each time specified EO classes have been instantiated. ***Linking on demand*** lets the system process all the EO instances within a product model and perform all the pertinent actions defined in UMEO.

Chapter 13 Application Scenarios and Software Prototypes

This chapter discusses issues arising in the context of practical implementation and usage of the above-mentioned architectures. A scenario-based approach will be motivated and the corresponding software prototypes elucidated. Further, certain technical implementation aspects will be covered. Although all the scenarios will be described and discussed following a uniform structure, not each issue will be taken as a topic for each prototype. This is due to the differing relevance of the issues and also due to spatial restrictions of this thesis. Also due to space constraints, many of the detailed results and insights gained during this research cannot even be hinted at below.

13.1 Motivation to Use Scenarios and Prototypes

The question of why to use scenarios at all will be answered during this section.

In theory, the ULEO approach is universally applicable (at least) in the entire field of product development. General practical applicability of a scientific approach, however, can hardly be proved – and is not necessarily demanded. Instead, it seems reasonable to judge an approach by its ability to yield substantial progress and benefits in any single application domain of sufficient relevance. Hence, several crucial areas within automotive product development at a major OEM are described in the following. This OEM's IT environments are subject to essential improvements concerning process integration and user support and thus are ideal application fields for the ULEO approach. Based on experiences gathered herein, a more general applicability of the ULEO approach can be estimated. A **scenario**, in this sense, is a collection of subsequent tasks to be performed in order to achieve certain milestones within the overall product development process. These tasks are performed by engineers on the basis of certain product information and rely on a given IT environment. Scenarios are not only used to discuss technical details, but also to show up the new way of work from the engineers' point of view.

☞ The scenarios discussed here describe parts of the new and desired way of product development, rather than the current situation.

Even though automotive engineers assisted in the ULEO approach's development by giving practical information about their work and feedback to draft-versioned demonstrators, **prototypes** are an essential means of showing usability, revealing problems and further application fields in practice. Future users can be made familiar with new concepts and incited to give valuable comments. The **ULEO prototypes** in part use the base functionality of commercial ProSAps and extend it by additional functionality within the discussed scope. Thus, these prototypes are not intended to serve as workbenches designed for integration into several other systems such as Shah et al.'s A.S.U. features test bed (see *[Shah et al., 1990]*), but offer specialized functionality assumed from real-world situations instead.

✍ The prototypes described in this section have (also) been installed within the **Digital Engineering Competence Center (DECC)**, which is a software laboratory equipped with the technical infrastructure to embed software prototypes into a realistic environment of scenarios from the domain of product development. As this infrastructure comprises also means for presentation, demonstration and discussion, the various ULEO prototypes and their underlying philosophy have been discussed with end users, IT staff responsible and scientists.

13.2 Scenarios and Specialized Prototypes for Design Automation and Annotation Management

The current section describes three early ULEO prototypes that implement the reduced MEOC and EOCL architectures introduced in Chapter 12.

While the *Ejector* scenario is realized with a MEOC prototype, EOCL prototypes support the scenarios C*ylinder head/crankcase bolting* and *Hole with boss*. All these prototypes have been developed in order to have a more plastic basis for discussing new possibilities of informational integration and automation with end users (engineers) in early phases of this research. They could be implemented with limited effort, as will be shown below.

13.2.1 Ejector of Cylinder Head Mold and Die Tools

This section's MEOC prototype was the very first of several prototypes developed in the context of this work. For details not covered by this section please refer to [Ourari, 2001].

Scenario and Scope of the Prototype

Scenario motivation – general. This scenario relates to the *multi-modeling* technique discussed in the section *Multi-Modeling Technique and Vertical Associativity between CAD Models* (page 20). It illustrates how engineering object constellations, purely implemented by means of commercial software, can support reuse and (in this case) uni-directional associativity to make routine work more effective and less error-prone. This relates to new and modified designs. EO constellations are building blocks in the above sense, consisting of two or more EO classes related to each other. In this simple version, however, the relation is not represented by independent relation entities but by formulas inside the individual EOs. The EO constellation's individual elements are bound for several product and resource models, thus integrating detail design and mold and die tooling on a low level.

Scenario motivation – what is new? Referring to the current state of product development, it adds task-specific EO classes (building-blocks) that are associated uni-directionally, thus avoiding errors occurring today on instantiation of isolated elements and allowing the engineer to move all the related EOs by moving a single central EO instance in the base model. Moreover, in spite of the just uni-directional associativity, one-way horizontal associativity is introduced between engine and mold and die tool design. It must be emphasized that this EO constellation relates features with non-features, which exceeds the usual limits of classical feature mapping.

Description of the prototype's functionality. Ejectors are components of mold and die tools, responsible for removing the freshly cast rough-part from the mold. They resemble screws or bolts, but possess a cascaded, decreasing diameter from their head to their tip.

The EO constellation *Ejector* covers features and parts to be used for the following product (partial-)models: cylinder head finish-part and rough-part, mold and die tool finish part, and machining model. To be exact, it provides building blocks for instantiating an ejector pin and its corresponding holes in the mold tools, as well as the contact bodies as part of the rough cylinder head, and the machining features inside the cylinder head's machining model.

User's View of the Prototype

The mold and die tool designer or the cylinder head designer may apply this EO constellation by instantiating the complete constellation inside the base model after having chosen it amongst several others by using a menu. These base model instances are correlated to each other by formulas inside their attributes and additional global parameters in the CAD model. One of these instances plays the role of a central reference object. The next steps are to copy the constellation’s individual elements from within the base model and paste them “with link”* into their respective destination models. If one or all of the constellation’s elements have to be changed, this is done inside the base model by changing the attribute values or the position of the central object. After the update of the destination models, all the elements of the EO constellation are again synchronized.

Discussion and Possible Enhancements

Drawbacks of this solution are obvious. Nevertheless, it is an improvement on the current state of engineering, indicating the chances of a better integration of product models and the development of new building blocks for the engineers. Accordingly, the benefits of applying the EO constellation *Ejector* in daily work were ranked very high by both M&D and engine designers.

13.2.2 Cylinder Head / Crankcase Bolting

The EOCL prototypes set out in this and the next section, have been developed to gain quick and general experiences based on an EO constellation linking architecture and to demonstrate its significantly more powerful functionality, achievable if deviating from the pure (untouched) current commercial software. The prototype that is in the focus of the next section follows the same principles but targets another application field. For details not covered by the two sections please refer to [Ananthanarayanan & Addala, 2002].

Scenario and Scope of the Prototype

Scenario motivation – general. Both prototypes implement the instantiation, editing, and deletion of EO constellations and include a simple annotation service. Compared to the MEOC prototype, they offer more automation and bi-directional associativity (dedicated EO relation entities are provided). So, while the MEOC prototype reduces the designer’s manual work and errors, the EOCL prototypes do the same to a significantly higher degree.

☞ Bi-directional associativity reduces many manual steps needed in case of uni-directional associativity.

The *Cylinder head / crankcase bolting* EOC implements the finish part aspects of what is commonly called an ***assembly feature (AF)*** functionality (see Figure 33): it concentrates all information relevant for instantiation and manipulation of an assembly situation given by of two or more individual EOs and respective destination product parts. Thus, it covers the most complex aspects of assembly features.

✍ The EOCL prototypes were not meant to be productively applicable, which is the reason for choosing quick and simple implementation means.

Scenario motivation – what is new? The EOCL prototypes offer all of the MEOC prototype’s new functionality, but add bi- and in principle multi-directional associativity, which facilitates editing of any element of the EOC. The annotation service, although of basic kind, can effectively support

Footnotes

* CATIA^{TM} terminology: a uni-directional link is created between the original and the copy

cooperation of engineers by providing meta-information on the design elements (EOs). Additionally, a flexible assembly feature solution is demonstrated.

Description of the prototypes' functionality. Embedded into an assembly feature instantiation and edition functionality, the *Cylinder head / crankcase bolting* EO constellation allows an engine designer to instantiate a bolted connection (i.e., one specific assembly feature) between cylinder head and crankcase in one action and with components spread over several CAD models. An assembly feature instance is an engineering object relation instance[*], carrying assembly-specific information and referring to an EO constellation instance holding the finish-part aspects of the assembly. Before instantiating the assembly feature, the engineer has to create the geometrical reference elements within the CAD. Having informed the user about the automatic insertions caused by the AF, the system creates holes through both finish parts (two individual CAD models), and instantiates the bolt within the assembly model and the bolted connection- (BC-) and AF instances within the EORI database. Synchronization of all the elements' attribute values is maintained. After the instantiation is finished, the user may select any of the EO instances in any model, or the assembly feature itself, for editing and attribute changes. Reacting on this, the system will inform users about these changes' effects on other EO instances and synchronize their attributes' values. Annotation functionality is offered, permitting users to add annotations (e.g., experiences or substantiations) to each element of the EO constellation. If any of the other models is opened by an engineer, this user is informed about the new annotation on one of its EO instances.

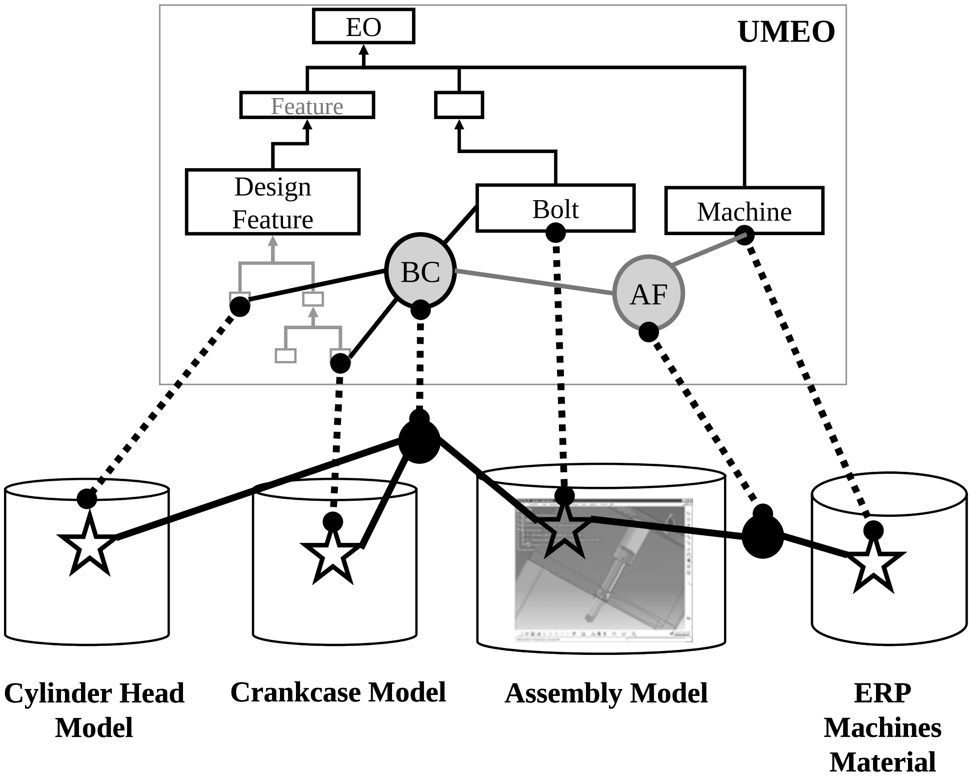


Figure 33: Assembly Feature Based on EO Constellations[†]

Footnotes

[*] …and thus no typical feature

[†] Acronyms: BC = Bolting Connection; AF = Assembly Feature

Technical Details

For the given reasons, a simple solution has been chosen for realization: the Visual Basic™ scripting language has again proved to be easy to learn and use and to allow building a prototype to demonstrate the functionality in a short period (see *[Ananthanarayanan & Addala, 2002])*. Therefore, automation and user interaction within the CAD system CATIA™ V5 have been implemented using Visual Basic scripts, the EOR instance database is represented by a Microsoft Excel™ sheet, acting as a relational database. EOR instances are stored in three tables: the *EORpartners* table stores the EOR partners linked by the respective bolting EO constellation instances, and the *EOCcontents* table holds the mathematical formulas correlating the respective partners' attributes as well as the IDs and locations of partners (EO instances). The *AssemblyFeatures* table holds the assembly feature instances, instantiated by this prototype and referring to the EOC instances stored within the other tables.

For further illustration, the fields of the ***EORpartners* table** fields are listed below:

~ Unique ID of the EO constellation instance (i.e., of the bolting EORI)
~ Information on partners (same for a maximum of 5 partners[*]):
    - CAD model[†] name of the first EO partner (complete file path)
    - ID of the first EO partner (complete path, referring to the specification tree inside the CAD model)
~ ID of the *Driving Feature* (complete path inside the CAD model)
~ AssemblyFileName (CAD model[‡], complete file path)
~ CAD model[§] name of the respective Assembly Feature (complete file path)
~ ID of the Assembly Feature EORI

The *driving feature* is optional and identifies the feature or EO instance whose attributes have to be assigned values by the user on instantiation. The other EOC partners' attributes' contents are calculated by using the formulas in the *EOCcontents* table. If no driving feature is specified, any partner's attributes may be given first by the user.

The *AssemblyFileName* specifies the CAD model, in which the bolt has been instantiated. The assembly feature contents refer to the *AssemblyFeature* table, explained below.

The ***EOCcontents* table** holds the following fields:

~ Unique ID of the EO constellation instance (i.e., of the EORI)
~ Information on partners to correlate (same for a maximum of 5 partners[**]):
    - CAD model[††] name of the first EO partner (complete file path)
    - ID of the first EO partner (complete path, referring to the specification tree inside the CAD model)
~ Formula, mathematically correlating the specified partners.

Each record stored within the ***AssemblyFeature* table** consists of the AF's identifier, the related EOC (bolted connection EORI instance), and further assembly-specific information on resources such as

Footnotes

[*] Unnormalized database structure
[†] .CATpart
[‡] .CATproduct
[§] .CATpart
[**] Unnormalized database structure
[††] .CATpart

machining costs, machining time. This additional information is not really deployed in the current implementation, however.

Two other tables serve as data basis for the annotation service: the ***annotation recipient*** **table** holds information on which annotations are to be displayed for which EO instances and the respective display status. The ***annotation text*** **table** holds the annotation contents together with the EO instance IDs they have been attached to. The principle: if an annotation is attached to a partner of a given EOC instance, this information is stored in the annotation text table. In addition, a record is inserted into the annotation recipients table for each other partner of the given EOC instance. If an annotation has been displayed, the respective record is marked accordingly.

After activation by the user, a Visual Basic script displays dialogs for retrieving the AF type and attributes as well as the driving feature's attributes and informs the user about the consequences of the offered automation. It finally performs the automatic instantiation by creating holes through both finish parts and by instantiating the bolt within the assembly model. In this version of the prototype, all the EOs are hard-coded.

Some **experiences** made during the practical implementation shall be mentioned in the following:

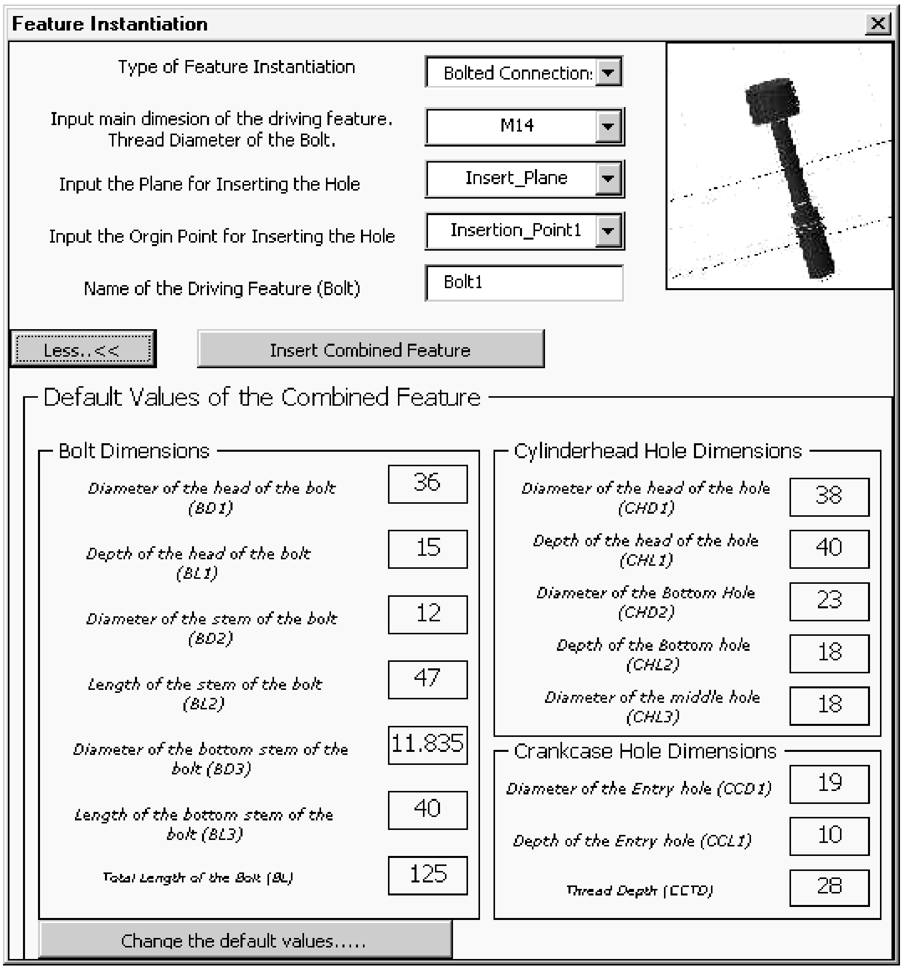
- From within Visual Basic scripts, user interaction regarding the CAD model is not supported; it is not possible to select elements in the design model.
- Retrieval of feature IDs constituted a problem caused by an immature API state.
- Online documentation of the APIs was insufficient, e.g., in respect to feature instantiation or the meaning of error messages.
- There is an annotation functionality of CATIA^{TM} V5 supporting the attachment of texts or files to a feature. Although helpful in some cases, it does not serve the purpose of automatic information exchange between designers.

User's View of the Prototype

From the engineer's view, this scenario is about creating (instantiating) and modifying assembly features for fixing a cylinder head on a crankcase using a bolt. As cylinder head CAD models belong to the most complex amongst all CAD models in the automotive area, real models slow down a CAD system significantly. For that reason, simplified CAD models were created for test purposes. By pressing a button, the designer invokes the assembly feature instantiation dialog allowing the user to select a desired AF type (Figure 34).

The current prototype does not use UMEO, but reflects the respective contents in its hard code. Now, the user can specify the main dimensions of the bolted connection from which the other EOs' attribute values are calculated by the Visual Basic script and presented as editable default values within the expanded user dialog.

After pressing the ***Insert*** button, another dialog is brought up retrieving geometrical input elements of the individual component features. Subsequently, the macro creates the assembly feature and all the EOs comprised inside the various CAD models, and including all formulas and rules, stored within them to correlate their attributes based on CATIA^{TM} functionality (the result is illustrated in Figure 35).



Feature Instantiation

Type of Feature Instantiation: Bolted Connection:

Input main dimesion of the driving feature. Thread Diameter of the Bolt.: M14

Input the Plane for Inserting the Hole: Insert_Plane

Input the Orgin Point for Inserting the Hole: Insertion_Point1

Name of the Driving Feature (Bolt): Bolt1

Default Values of the Combined Feature

Bolt Dimensions

Diameter of the head of the bolt (BD1)	36
Depth of the head of the bolt (BL1)	15
Diameter of the stem of the bolt (BD2)	12
Length of the stem of the bolt (BL2)	47
Diameter of the bottom stem of the bolt (BD3)	11.835
Length of the bottom stem of the bolt (BL3)	40
Total Length of the Bolt (BL)	125

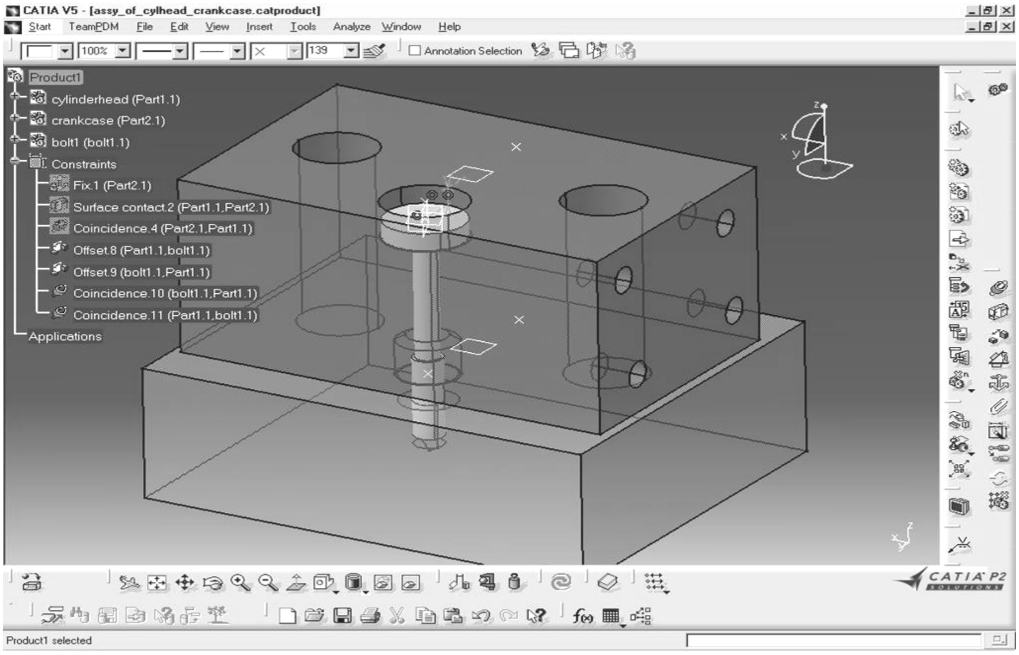
Cylinderhead Hole Dimensions

Diameter of the head of the hole (CHD1)	38
Depth of the head of the hole (CHL1)	40
Diameter of the Bottom Hole (CHD2)	23
Depth of the Bottom hole (CHL2)	18
Diameter of the middle hole (CHL3)	18

Crankcase Hole Dimensions

Diameter of the Entry hole (CCD1)	19
Depth of the Entry hole (CCL1)	10
Thread Depth (CCTD)	28

Figure 34: Instantiation Dialog for Assembly Feature EOC [Ananthanarayanan & Addala, 2002].



CATIA V5 - [assy_of_cylhead_crankcase.catproduct]
Start TeamPDM File Edit View Insert Tools Analyze Window Help
Annotation Selection
Product1
cylinderhead (Part1.1)
crankcase (Part2.1)
bolt1 (bolt1.1)
Constraints
Fix.1 (Part2.1)
Surface contact.2 (Part1.1,Part2.1)
Coincidence.4 (Part2.1,Part1.1)
Offset.8 (Part1.1,bolt1.1)
Offset.9 (bolt1.1,Part1.1)
Coincidence.10 (bolt1.1,Part1.1)
Coincidence.11 (Part1.1,bolt1.1)
Applications
CATIA P2
Product1 selected

Figure 35: Instantiated Assembly Feature [Ananthanarayanan & Addala, 2002].

Within the assembly feature ***edit*** dialog, a button can be pressed to invoke the annotation functionality. Thus, the modifications performed within the edit action can be documented and handed over to engineers responsible for the respective CAD models the AF spans over (cylinder head model, crankcase model, assembly model). When one of these CAD models is opened, the users can check for the presence of annotations by pressing a button and subsequently view the annotations.

Discussion and Possible Enhancements

The prototype described is suitable to serve as a vivid base for discussions with domain experts, when practical applicability and acceptance of AF and EOC concepts based on ULEO are to be illuminated. Also, GUI variants can be compared and evaluated. It thus contributes to assessing the principle concepts.

Nevertheless, EOC linking is not only restricted to and promising for prototypes. Based on a more solid implementation platform, this functionality can be productively deployed.

As the prototype's functionality has been limited by specification, it can be expanded in several ways, when EOCL is to be applied practically. The most significant improvement is to retrieve the now hard-coded EO class descriptions at runtime from some implementation of the UMEO database, in the simplest form represented by a ULEO XML file. Thus, the AF functionality would be generalized to <u>any</u> kind of AF, while the set of supported assembly features would be changeable by modification of the XML file.

Concerning software implementation, the restriction of Visual Basic scripts in conjunction with CATIA^{TM} V5 suggests that the less restrictive, but more demanding CAA (C++) API should be used; it requires dedicated licenses and programming skills and, additionally, extensive experience in using the API effectively.

13.2.3 Hole with Boss

This section describes a prototype that is very similar to the one described in the preceding section. However, the application scenario changes. EOCL is applied stand-alone (without AF context) to support MML model handling of cylinder heads (see also section Multi-Modeling Technique and Vertical Associativity between CAD Models on page 20). Due to the common underlying technology and to avoid redundancy, this prototype will be described more concisely.

Scenario and Scope of the Prototype

Scenario motivation – general. The current scenario uses the EO constellation* *Hole with boss* to cross-link different parts within a multi-model environment by offering multi-directional associativity: rough-part model, machining-model and finish-part model.

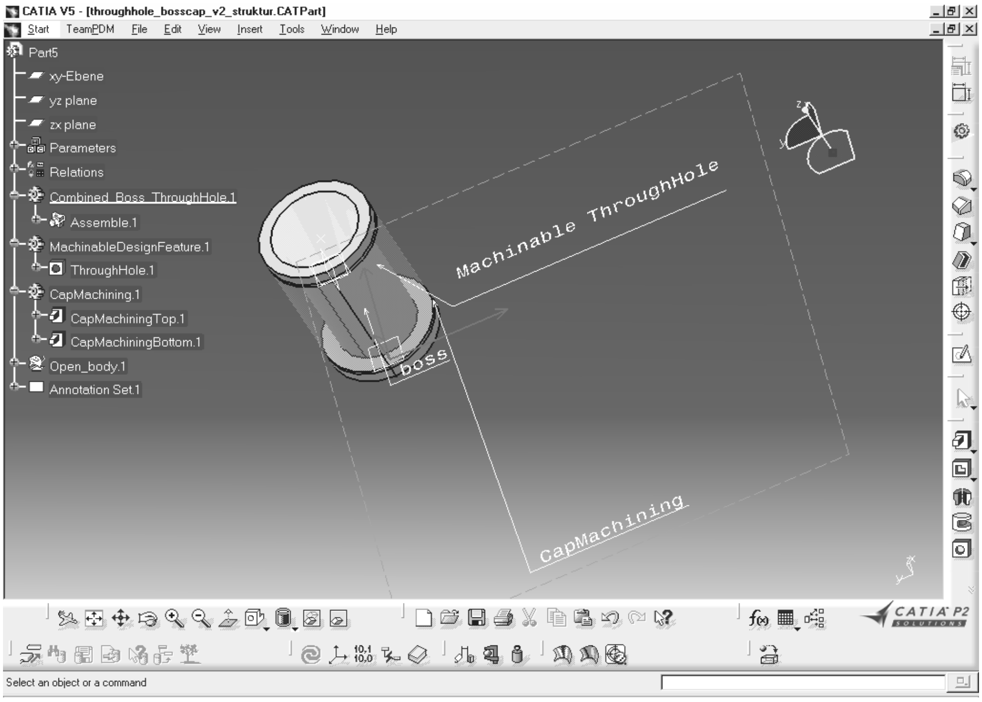
Scenario motivation – what is new? Multi-directional associativity is achieved here as a further improvement to the current state in the productive product development (see again section *Multi-Modeling Technique and Vertical Associativity between CAD Models* on page 20). Also in this rather simple case, quite manifold relations exist between EOs, as Figure 41 illustrates.

Description of the prototypes functionality. This EOCL prototype implements the instantiation, editing, and deletion of a feature constellation (FC), consisting of finish-part feature, rough-part feature and machining-feature, and offers the above-described annotation service. The finish-part feature is a (negative) *hole* feature. The corresponding rough-part feature is a positive *cylinder* that is

Footnotes

* To be more precise, <u>feature</u> constellation

added to the rough-part to assure minimal wall-thickness around the hole. If placed close enough to the cylinder heads boarders, it produces a boss surrounding the hole. Moreover, the rough-part feature contains additional material, covering the final hole. This material is part of a *thickness*, which has to be removed from the rough-part before the hole is drilled (see Figure 36 for an illustration of these elements). As this scenario is based on the MML methodology, the machining feature is also positively adding material to the machining model.



Select an object or a command

Figure 36: Feature Constellation Hole with Boss [Ananthanarayanan & Addala, 2002].

Technical Details

The FC *Hole with boss* uses hard-coded feature types, as applied for the EOC *Cylinder head/crankcase bolting*. Logically, it realizes the UMEO contents, depicted in Figure 37.

Practical experiences. As described in the section on multi-modeling and in the description of this prototype above, the machining model contains positive geometry going to be subtracted from the rough-part model. However, CATIA^{TM} V5’s programming interface doesn’t support the creation of positive holes in a new *body*. This problem was solved by using a positive *pad* feature instead of a hole. For more details on the implementation of the prototype, please refer to the previously described prototype.

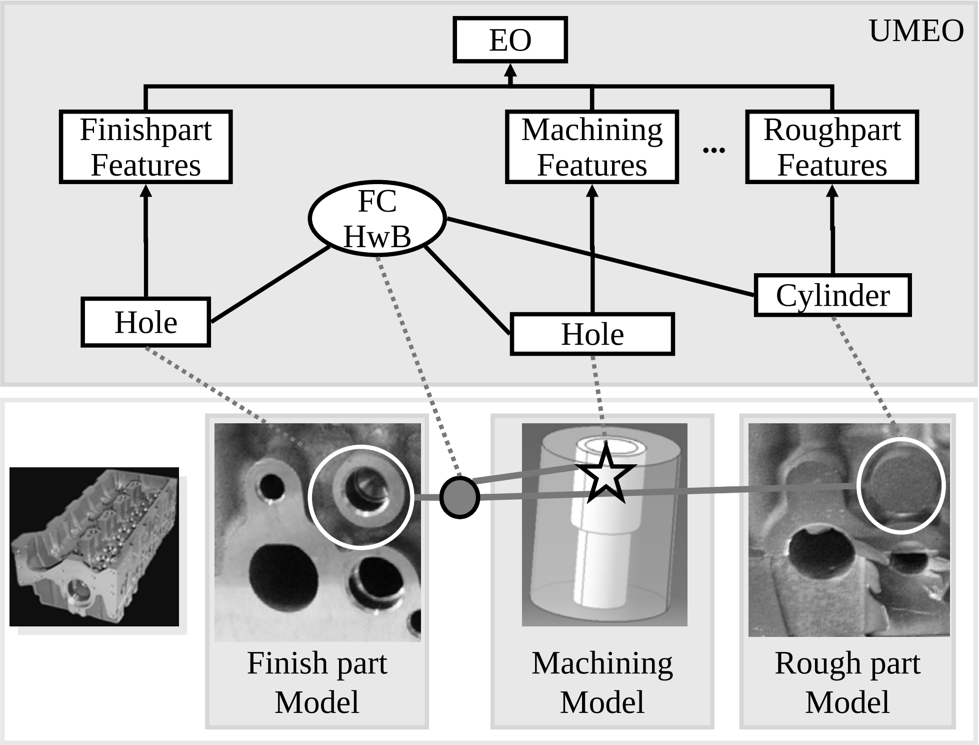


Figure 37: Feature Constellation Hole with Boss*; schematic*

User’s View of the Prototype

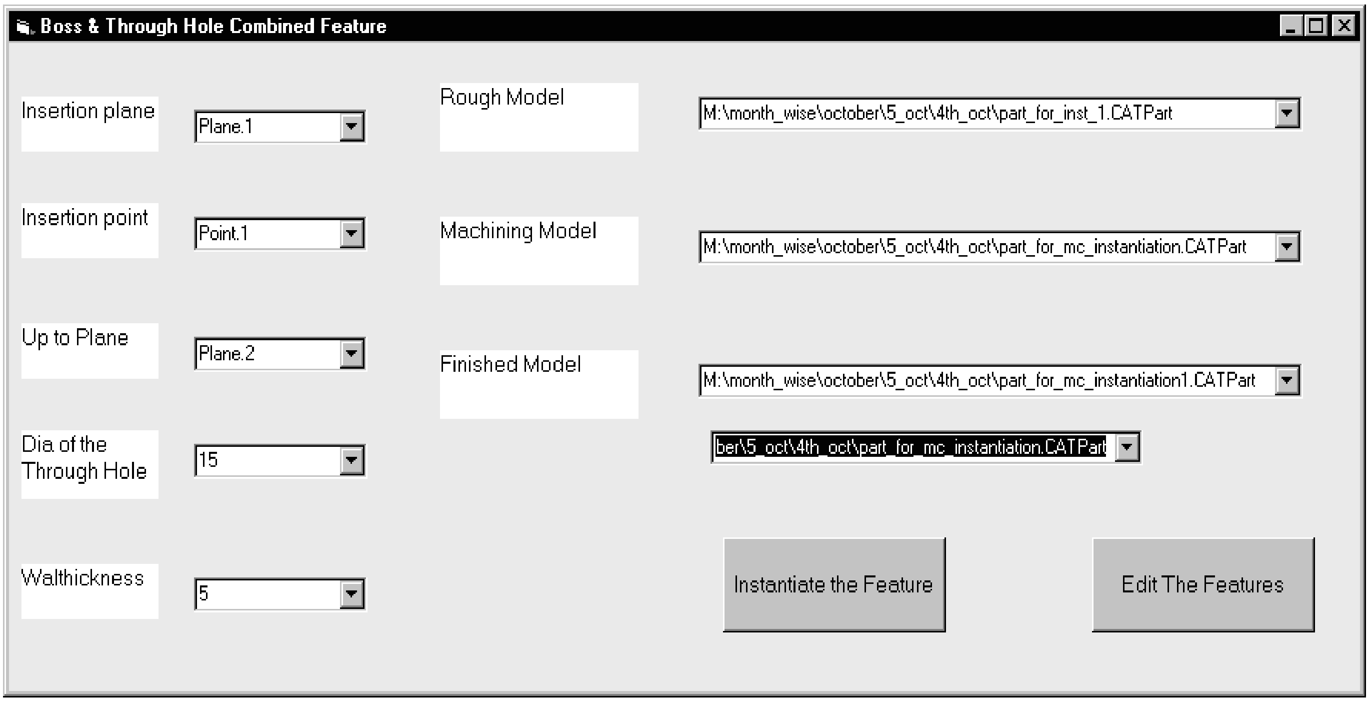
By pressing a button, the instantiation script is invoked. A dialog comes up, allowing the engineer to specify reference geometries and respective destination models for each feature in the constellation (see Figure 38). Also the attributes of the finish-part feature have to be specified.

Triggered by the *Instantiate* button, the script opens the specified CAD models and instantiates the respective features inside them. After instantiation, any of these individual feature instances can be selected for edition. The designer will be again informed about the parts affected by the running modification. After confirming the modifications, all changes will be propagated to the feature instances within the other CAD models.

Discussion and Possible Enhancements

Basically, what has been discussed in the previous section is also applicable to this prototype.

The *Hole with boss* scenario turned out to be a frequently occurring situation in an engine designer’s daily work. As a consequence, the IT staff responsible for cylinder head design has decided to support designers in this respect. However, due to their policy of sticking to standard (unmodified) CAD functionality, they implemented an MFC solution for *Holes with bosses* using CATIA^{TM} *PowerCopies* and libraries. As will be explained below, other IT managers living in a different IT landscape decided differently. It can be observed here that the chances of IT concepts of being implemented depend also on marketing models of IT departments.


Boss & Through Hole Combined Feature

Insertion plane: Plane.1

Insertion point: Point.1

Up to Plane: Plane.2

Dia of the Through Hole: 15

Walthickness: 5

Rough Model: M:\month_wise\october\5_oct\4th_oct\part_for_inst_1.CATPart

Machining Model: M:\month_wise\october\5_oct\4th_oct\part_for_mc_instantiation.CATPart

Finished Model: M:\month_wise\october\5_oct\4th_oct\part_for_mc_instantiation1.CATPart

ber\5_oct\4th_oct\part_for_mc_instantiation.CATPart

Figure 38: Instantiation and Edit Dialog of Hole with Boss *FC*

13.3 Universal Prototype and Short-Term Application Scenarios

This section targets a (single) ULEO server prototype and surrounding applications mirroring the above-proposed full-service ULEO architecture. This ULEO server is universally applicable to many practical application scenarios. Subsequently, the scenarios that it has already been applied in or that are candidates for application in a short-term period of time are described.

The universal ULEO server prototype described in this section has been and is still being incrementally implemented according to the functionality needed within the individual scenarios. As all the ULEO applications involved base on the unrestricted ULEO architecture, they will incrementally be suitable for providing and evaluating all ULEO features.

13.3.1 Quality Assurance for Sheet Metal

This scenario describes an application of the ULEO approach that has already been chosen for application in productive car development. For this purpose, the ULEO server prototype has been enhanced to reach the state of a productively applicable software application.

The productive Version of the ULEO server has been applied in a pilot phase until the beginning of the year 2005. Since finishing of the pilot phase, it is being productively applied in the quality assurance for Mercedes cars.

Scenario and Scope of the Prototype

Scenario motivation – general. The application starts off tackling certain quality assurance (QA) aspects as described in the following. This scenario will be referred to as *quality assurance scenario* or concisely *inspection scenario*. The motivation for applying ULEO coincides with this research targets.

Scenario motivation – what is new? Consequently, this software implementation offers a large range of new possibilities, coming with the implementation of a global information space and means for re-use and automation. The ULEO concepts and software are currently applied in the domain of quality assurance as the first step toward informational integration of the following applications: **CAD system** CATIA™ Version V[*], **inspection planning system**[†] **MPE**[‡] based on CATIA V5 CAA APIs and **inspection programming systems** of other software manufacturers (DELMIA Inspect™, CIMstation™, Holos NT™, Metrolog™ V5, Calypso™, etc.) and the **tolerance analysis software** 3-DCS™. Primarily, "informational integration" means here to facilitate the information flow from proprietary data sources (CAD, inspection planning tool, tolerance analysis tool) into a neutral data pool (ULEO database) granting access to any software application, and to share the respective background knowledge comprising the concept- and relation classes involved and including the machine-interpretable strategies employed in the quality assurance process[§].

☆ For instance, the inspection strategies are retrieved or interpreted at runtime by the inspection planning tool and an inspection programming application. Figure 39 illustrates three alternative inspection strategies for a slot.

Facilitating the information flow includes promoting multi-directional associativity between informational entities in order to, for example, enable automatic detection of changes and further support to change management. The downward flow of product function information and upward flow of aggregated inspection results contribute to the same issue. Figure 40 shows the rough architecture of this scenario.

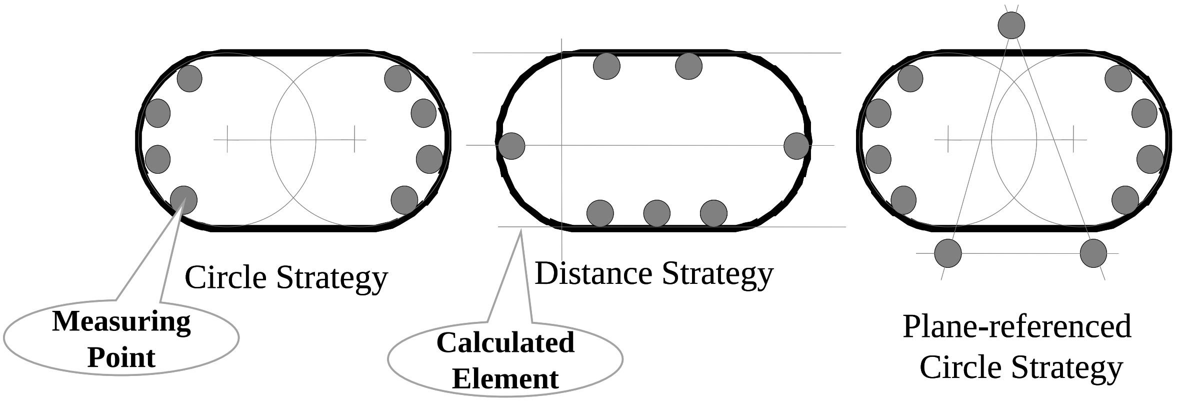


Figure 39: Various Inspection Strategies for a Slot [Th. Karthe]

Footnotes

[*] Assemblies of finish-parts, tolerancing information, clamping concept

[†] Quality criteria, inspection planning elements, partly measuring elements

[‡] German *Meßplaneditor*

[§] Inspection strategies, analysis strategies, visualization strategies, etc.

- ✰ The designer will be able to access inspection results for any given finish-part design. This includes compressed and consolidated statistic information that is useful for reasoning about practical benefits or drawbacks of certain geometric solutions. Backward information flow from quality assurance back to the design is relevant, for instance, during the ramp up phase of a new car product series, when specific design solutions are weighed up against each other; such solutions may vary in terms of matching tolerances better or worse, or of different kinds of flanges producing more or less tensions. As product function modeling is also about to be integrated in the real product development process, the designer will be able to retrieve inspection results function-oriented. Pertinent engineering experiences will also be available from within UMEO, formulated as generally valid know-how after having been entered by a dedicated expert group.

The information flow between inspection planning and tolerance analysis tool allows inspection and tolerance simulation to share inspection points that are of common relevance.

A new ***inspection strategy assistant (ISA)*** tool will allow experts to create and modify script-based algorithms for deducing sets of measuring element instances from quality criteria. Measuring elements are to be interpreted by a coordinate measuring machine. During the information modeling work within the I++ work group (see section *Inspection-plusplus* on page 183), quality criteria have been identified to be relationships between geometric objects and tolerance information (see also *Figure 4: Impression of the I++ Information Mode*).

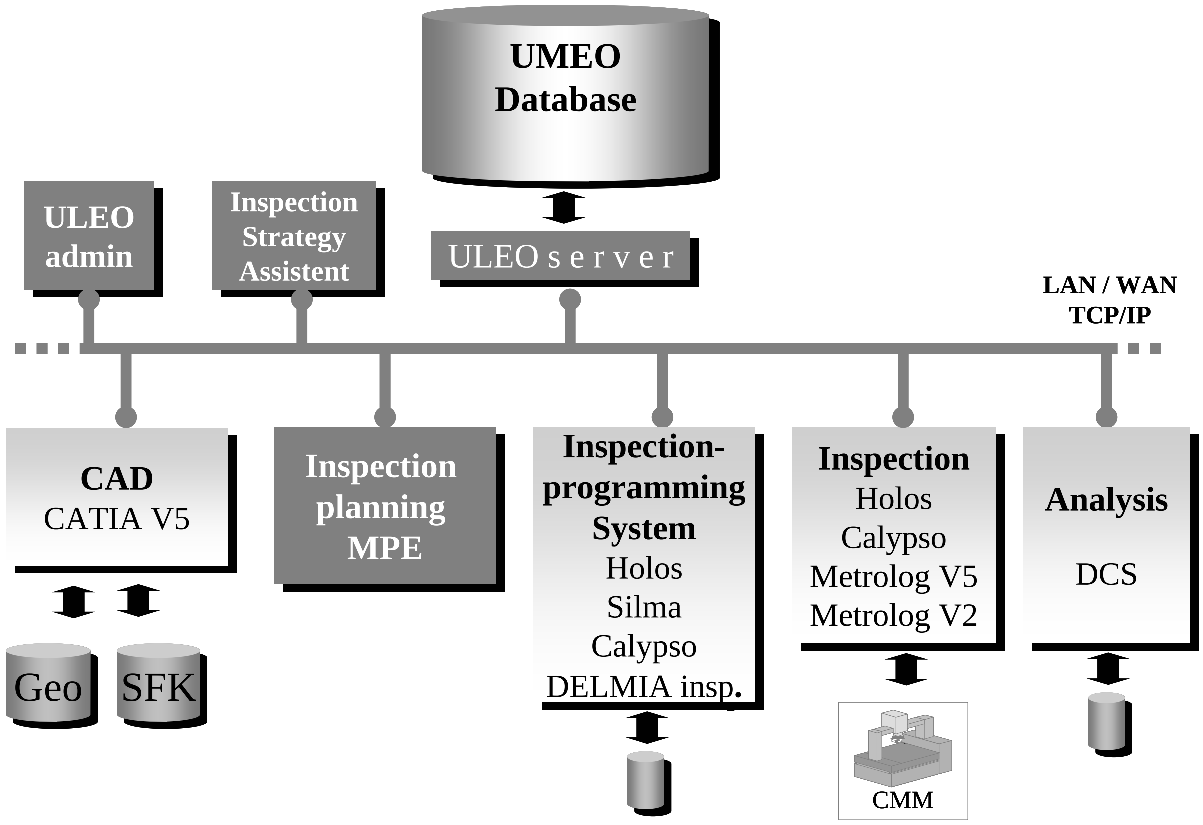

UMEO Database
ULEO admin
Inspection Strategy Assistent
ULEO s e r v e r
LAN / WAN TCP/IP
CAD CATIA V5
Inspection planning MPE
Inspection-programming System Holos Silma Calypso DELMIA insp.
Inspection Holos Calypso Metrolog V5 Metrolog V2
Analysis DCS
Geo
SFK
CMM

Figure 40: Inspection Scenario – Applications Involved[*]

Footnotes

[*] Boxes in light color depict pre-existing commercial applications

Technical Details

A ULEO server gives the above-mentioned applications access to UMEO, MTRT and the EOR Instance database and partly handles their EO instance management. The inspection planning tool stores its data completely inside the ULEO database. The tolerance analysis tool uses its own information management but writes tolerance analysis points into the EO instance database. EOR instances are used to inter-connect the respective applications' instance information multi-directionally.

Implemented Software Applications

The ULEO implementation comprises several software applications: a **ULEO server** as specified in Chapter 11 *The Proposed Software Architecture*, an **administration client**, subsequently also denoted as *ULEO admin* and the (not yet finished) **inspection strategy assistant** *ISA*. On the side of productive applications, this scenario is completed by the mentioned **inspection planner** *MPE* and the **tolerance analysis tool** (see Figure 40). A prototypical customization of the **inspection programming** tool *DELMIA inspect*™ has also been realized. Of course, this set of integrated applications sharing a global information space is intended to be extended. From the current viewpoint, a **supplier integration client** will be one of the next candidates to be implemented, as there are strong demands for it from the engineers responsible. For details on ULEO server and ULEO admin please refer to *[Mellens, 2005]*.

ULEO Server

The ULEO server comprises several main components (see also Figure 31) realizing the core functionality: in addition to a **UMEO XML parser**, **UMEO XML templates** are used to efficiently create new EOx objects; they avoid building them up character-wise. **Server add-ons** are modules that are tightly integrated (linked) into the ULEO server, and therefore not communicating via the IPCI. The EXPRESS converter described below in this section is such a server add-on. **Logging** of the ULEO server's actions, of occurring errors and of client access happens into files, to the console or to other clients (remote). Basic **configuration** can be specified via a configuration file. Dedicated modules handle the **inter-process communication** thus spawned as dedicated tasks in parallel. The **DB connection pool** manages database connections efficiently and avoids, for example, frequent execution of time-consuming *connect* and *disconnect* actions. An **XML/database converter** is able to write all UMEO/MTRT contents to a ULEO XML file and back (see also *[Van den Elst, 2002]* for the first trial release). This feature can be used for matters of archiving and versioning and also to hand over instance information via files.

File-based information delivery may result in practice, when administrative structures ban online-cooperation of software applications. This effect can be regarded as step in the evolution toward the final target of global information space, as this administrative restriction may in turn result from technical restrictions.

RAM caching is performed for UMEO, MTRT and for such EOx instances that are handled by the ULEO server (EO(R) instance database). Also the management of sessions and users logged on to the ULEO server, as well as the services offered by them, is kept in the RAM. As individual EOx entities do not require much storage space (typically around 200 to 1024 bytes[*]), current computers are able to keep large parts of these sorts of information in their memory.

Footnotes

[*] Not limited

While the ULEO server prototype has been developed on the Windows 2000^{TM} operating system, the productive version runs on a UNIX system operated by an IT service department.

ULEO server add-on *Express Converter*. In the context of this work, a three months study has been performed by Eric Narby (see *[Narby, 2003]*) to investigate the possibilities of make STEP-based files available from within the global information space. For this purpose, an experimental prototype has been developed, called ***Express2UMEO***.

Express2UMEO translates information models written in the STEP representation formalism EXPRESS (ISO 10303-11), into ULEO XML format and stores the results inside a respective XML file. Such files may then be imported into UMEO. Although not implemented yet, the Express2UMEO conversion will be integrated into the ULEO server application and made accessible as an administration service for the administration client.

The ultimate intension is to import various domain-specific EXPRESS models as sub-taxonomies into UMEO and to manually correlate them to other UMEO contents using EORMs in order to reflect the respective semantics. On this basis, STEP product data files adhering to the EXPRESS data models imported may be interpreted and used by any ProSAp.

During the conversion, EXPRESS expressions are mapped onto the ULEO information structure. During this conversion, certain EXPRESS data types are mapped n:1 onto ULEO standard atomic data types (XML schema types, as discussed above). Complex data types are published as independent data types in UMEO, entered into the respective sub-taxonomy and referenced from within the Express EO classes.

No information gets lost during the conversion to ULEO XML; information considered to be EXPRESS-specific is included inside string-typed EO attributes or methods as *abstract syntax trees (ASTs)*. The respective attributes' meta-information specifies the EXPRESS version. This matches the philosophy of hybrid knowledge representation within attributes and methods (see section *Details on Hybrid Information Representation within Attributes and Methods* on page 128 for details). Such information comprises functions and procedures (stored as methods), rules such as *WHERE* and *QUERY* clauses (stored in methods), *select* types and other expressions which cannot be interpreted by Express2UMEO.

Express2UMEO has been written in the *Java^{TM}* programming language. Java was chosen as the performing person's programming skills did not comprise *Python* and as this prototype's integration into the ULEO server could serve as scenario to test the effort necessary to integrate other-programming-language applications into a Python application.

Express2UMEO uses the *EXPRESS Open Source Parser** to parse EXPRESS code to an AST. It then walks the AST and outputs XML according to the ULEO XML format using the XML package *JDOM*.

System Requirements: Express2UMEO will run on any system with a *Java Virtual Machine (Java VM)* that supports at last Java 1.2.

Administration Client *ULEO admin*

The need for an administration component has already been stated above. In order to allow administration from any location in the network, the ULEO admin has been implemented as a ULEO GIS client interacting with the ULEO server via the GIS IPCI. As the implemented ULEO server already offers a rudimentary user management (users and rights to call individual GIS services) unauthorized access can be prohibited. In the current version, the ULEO admin is able to support direct manipulation of UMEO and MTRT as well as the EO instance and EOR instance databases.

Footnotes

* See *http://sourceforge.net/projects/osexpress/* for further information.

Furthermore it allows the software developers to assess the efficiency of network and database accesses. Management of users, licenses and access rights is currently being enhanced, consistency and redundancy checks for data base contents, as well as the viewing of logging results about system failures, and of access logging and statistics are currently being implemented. Not yet implemented are means for handling archiving and versioning.

An impression of the user interface is given in the section *ULEO Server and Administration Client* on page 184.

Client ProSAps

This section describes productive applications that have been integrated into the GIS.

Inspection Planning Editor *MPE*

The *MPE* has been developed to take over functionality that today is performed partly using CAD systems and partly by using inspection programming software. As these tools do not support sophisticated inspection planning and as they are highly expensive, the MPE helps to save costs for licenses while providing better functionality. The MPE is part of a global information space as described above; in order to communicate with the ULEO server, the MPE adheres to the ULEO IPC interface.

The MPE tool is implemented using the CAA C++ API and is a new CATIA^{TM} workbench, which means that it runs within the integrated CATIA environment sharing the look and feel with the other workbenches. Using library functions, it is able to access proprietary Dassault[*] files for accessing **input information** for inspection planning (finish part, tolerances, clamping and fixing concept). In order to realize the mentioned CATIA look and feel – which is important, as it looks familiar to the engineers – customized CATIA visualization elements are used and temporarily stored as CATIA data structures within an extended *CATproduct*-type[†] document. Although these documents are storable for temporarily saving the inspection planner's work, the produced **output information** is finally made persistent by creating EO(R) instances within the ULEO database. CAD files may be visualized within the MPE to support a comfortable inspection planning, but they are not really necessary. Once a certain inspection plan has been stored in the ULEO database, it can be retrieved again without reloading the CAD files. To assure backward compatibility to the older CAD system CATIA V4, also comma-separated csv files can be imported as input for the inspection planning. They contain feature and tolerance information and can be produced running CATIA V4 macros on the finish part. Likewise, and for the same reason of backward compatibility, csv files can be generated carrying the inspection planning results to today's inspection programming system *CIMstation* (see section *Quality Assurance for Body-in-White Parts* on page 26).

The first release of the MPE does not provide **GIS server** functionality, but this will be implemented for facilitating the cooperation with the above-mentioned tolerance planning system.

During the inspection planning, also inspection programming tasks can be performed, as has already been mentioned. Thus, **inspection strategies**, stored within EORMs, have to be retrieved from UMEO. To achieve this, the MPE software uses the ULEO IPCI and retrieves all EORMs of a given EOR type (*sheet metal inspection strategy*) that are correlated to that *Quality criterion*-EORM currently handled by the user within the MPE.

Footnotes

[*] Vendor of the CATIA CAx system

[†] Dassault document type for managing assemblies

✍ Please remember that quality criteria types are also EORMs and the inspection planning process starts with selecting quality criteria instances (EORIs) to be inspected. These EORIs are stored in the UMEO EOR instance database, while the respective EORMs are stored within UMEO.

Thus, all available inspection strategies for a given quality criterion can be retrieved efficiently, without getting undesired results from UMEO.

An impression of the **user interface** is given in the section *Inspection Planning Tool*.

Inspection Strategy Assistant

The *Inspection Strategy Assistant (ISA)* is a tool that is intended to be used for creating and editing **inspection strategies** and **analysis strategies** while providing a best possible user support. In *[Okeke, 2002]*, a first illustrator has been implemented using Visual Basic™. An impression of the user interface is given in the section *Inspection Strategy Assistant* on page 186. This tool is scheduled for re-implementation until the end of 2005. Currently it does not work with real data and has no ULEO IPCI functionality. The final release will allow the users (a group of experts responsible for company-wide standardization of inspection strategies) to **create** new **EOR types** for carrying inspection and analysis strategies, to materialize them into UMEO and to edit and remove them. A strategy will be stored within an EORM method in the language ***ISRL***[*].

While the current implementation is based on Visual Basic, the productive version will be implemented using Java or C++.

DELMIA Inspect Prototype

This prototype has been implemented during the diploma work of Mario Thome (see *[Thome, 2003]*) as a customization of the inspection programming tool *DELMIA Inspect*, which belongs to the family of Dassault products sharing a common look and feel with CATIA™[†].

✍ Generally, Dassault applications can be customized by two means: (a) so-called *automation* by scripting using Visual Basic or Dassault-specific *CATscript* or (b) *CAA coding* using C++. *Automation* is the faster option to implement, needing less programming skills but being more restricted concerning functionality available. For diploma theses, *CAA coding* is not feasible in general, as learning CAA usually takes about 3 months, the required profound C++ skills not included.

This prototype has been implemented to investigate and quickly demonstrate the functionality of the MPE tool, which has been developed afterwards. In this respect, the DELMIA prototype served also as **part of the MPE's specification**. On the other hand, it will also be used to perform inspection programming tasks **integrated** into the **new** ULEO **architecture**. In this scenario, the application of *DELMIA inspect* will be restricted to a rather simple task.

To support inspection planning tasks, specialized **user-defined features** have been created to visualize **quality criteria** (design features plus tolerances) and the **inspection strategies** and their parameters, chosen by the user. Inspection sequences can be specified using a table-based dialog.

Footnotes

[*] Inspection Strategy Representation Language

[†] DELMIA Inspect is a CAA application; DELMIA is a sub company of Dassault Systèmes.

Software Implementation

This section informs about the utilized tools and programming languages, and the experiences made.

The Python Programming Language

The ULEO server and its administration client (see section *Administration Client* on page 175) have been implemented prototypically in the context of the diploma thesis of Jannes Mellens (see *[Mellens, 2005]*) using the programming language *Python*[*] (see *[Raymond, 2000]*). The primary reason to choose Python amongst other current programming languages is that implementation time is a critical factor for prototypical software implementations. According to recent publications in the World-Wide Web – a comparison of Python to other programming languages can be accessed at *[Rossum, 1997]*) – Python is well-suited to meet the requirement of rapid prototyping and comes with several further features: integration into other applications, manifold libraries are available for standard purposes, ad-hoc testing via interpreter, compilability for improved runtime efficiency, clear and concise software code. The drawback of still sub-optimal runtime efficiency can be neutralized by re-implementation of critical code sequences using Java or C++. So, Python is advocated as a language suitable for designing applications rapidly and optimizable. Because of these properties, Python is also suggested as suitable for applications' main modules, while re-implementations are done for such modules requiring optimal runtime efficiency. This work's practical experiences confirmed this approach (see *[Mellens, 2005]*). Nevertheless, the principal decision, whether to chose Python or another programming language for practically used software, will certainly be located rather on the non-technical area and will be focused on available infra structure, preferences and regulations of the company's head office.

Tests of the prototypes indicated sufficient efficiency of the ULEO server and the administration client. Thus, it has been decided not to re-implement them for productive use but to mature the Python-coded base software.

Development Environments

Python comes with a software implementation environment, consisting of an editor, a simple debugger, a Python interpreter, a Python compiler and libraries for standard tasks such as GUIs and inter-process communication. In fact, these standard tasks were implementable in rather short periods of time (about two weeks for the *ULEO admin*'s GUI and one week for CORBA), although the GUI is not trivial and CORBA is commonly agreed to be powerful, but rather cumbersome to implement. Also *Web Services over http* are supported by a Python library. Although not as comfortable as, for example, Microsoft's Visual Studio^{TM}, the collection of Python tools turned out to support efficient coding.

The MPE client is developed using Dassault Systèmes' CAA development environment that supplements the Microsoft Visual Studio C++ and is therefore comfortable to handle.

ISA and the *DELMIA Inspect* prototype have been developed using the Visual Basic environment that supports coding well by comfortably offering necessary information; the debugging facility offered is rather basic, however, especially if compared to current C++ and Java environments.

Footnotes

[*] See homepage of Python Software Foundation at URL *http://www.python.org/psf/*.

Filling UMEO

This section covers the procedure and some details of gathering and modeling information for representation inside the UMEO database.

Process of Filling UMEO – Information Modeling

Overview. The section *Information Acquisition – Filling the Information Base* on page 118 suggests combining the top-down and bottom-up methods when acquiring and modeling background information to be stored within IIM/UMEO. While the first method tends to yield a rough but domain-spanning class taxonomy, the latter adds detailed and domain-specific information. As the top-down method critically depends on a sufficient degree of holistic thinking and a global overview, it is most effectively performed by heterogeneous groups encompassing experts from various domains rather than by single persons or homogeneous work groups. The range of tasks within product development that are ultimately to be covered by the IIM determines the exact interpretation of the notions "homogeneous" and "heterogeneous".

Accordingly, the currently available **UMEO contents** stem from a variety of **sources** that helped to assure a holistic body of thought, especially reflected by the UMEO taxonomy's upper layers and by practically useful IEs on detail level. The first (**top-down**) step was done by founding a research-internal information modeling work group, chaired by the author, and consisting of mechanical engineers and computer scientists, bringing in experience on the fields of powertrain design, powertrain manufacturing, and information modeling. Some of the workgroup members were actively involved in the STEP standardization efforts. This cooperation incrementally produced a first taxonomy of engineering objects, relevant in these areas (some more details will be given below).

Complementary detail work (**bottom-up**) has been done to produce the necessary domain-specific details: UMEO was incrementally supplemented, as during research projects and diploma works feature sets have been identified and implemented for practical application in design and quality assurance of powertrain and body in white. This taxonomy comprising specific feature sets was called ***Unified Feature Model (UFM)***. The currently existing UMEO hosts a superset of the UFM.

In parallel to the identification of new feature types, **feature standardization** activities were performed for a variety of powertrain hole features, finally culminating into a DIN standard (see *[DIN 32869-3-4]*) and a company-internal standard, extending the DIN features for specific in-house features. The motivation was to push the availability of standardized tools for machining of holes.

The feature- and other **EO classes** found for the given domains and represented in UMEO are ProSAp-specific in the sense that they were practically implemented using certain commercial software tools such as *CATIA*^{TM} V5 or *SILMA CIMstation*^{TM}. The corresponding EO class descriptions were inserted into UMEO and refer to the physical tool-specific implementations by external references (see also section *One-Entrance Principle and External Referencing* on page 110). CATIA-FTA-specific tolerance structures have been analyzed (*TTRS*[*] trees, see *[Clément et al., 1998]*), but not directly added to UMEO, as they are not being used directly in the implemented global information space, but translated by the ProSAp add-ons to another representation instead. Relevant **relationships** have been identified mainly looking at potential EO constellations, new kinds of multi-model links, and for inspection planning. Some details are given during the following sections.

Footnotes

[*] *Technologically and Topologically Related Surfaces (TTRS)* model. Note that Otto W. Salomons applied the TTRS theory in the FROOM system (see *[Salomons, 1995]*).

When UMEO had already reached a state of practical usability, the author joined the **I++ working group** (see section *Inspection-plusplus* on page 183, and *[Zimmermann, 2004]*) in order to promote the automotive-OEM-spanning standardization of parts of the upper layer IIM contents. Also the resulting I++ information model will be highlighted below.

Another bottom-up contribution is tackled in current OEM-internal research projects where **EBoK knowledge** is newly arranged and correlated to the already identified EOs and EORs in order to assist the engineers in using rather complicated new **product templates**.

The task of **acquiring** and **modeling** information was supported by several **software tools**: textual interview protocols have been produced using standard office tools. Fragments of information models have largely been documented using static structure diagrams from UML. However, no tool was accessible being able to fully support n-ary relationships[*]. *Rational Rose*™, *Innovator*™, and *Microsoft Visual Modeler*™ all have significant restrictions in this respect. As the best compromise, *Microsoft Visio 2000*™ was identified and most frequently used. Unfortunately, it is not possible to interchange UML models between Visio and the other tools mentioned. In cases, where UML fragments where to be published to a larger audience, also *Microsoft PowerPoint™* has been used. Since the availability of the ULEO administration client (see section *Administration Client* on page 175) it was possible to create and edit information models also comprising n-ary relations, but there is no UML user interface thus far and it is only usable if there is no ULEO server accessible. For the future, a flexible tool support is desired by implementing a UML/ULEO-XML conversion tool (see also section *Prototype Functionality to be Implemented* in the outlook on page 212); furthermore a virtual reality 3D editing tool is planned for naturally browsing and editing IIM contents. Schumann and Müller (see *[Schumann & Müller, 2004]*) show various attempts to tackle this issue.

Practical Experiences with Expert Interviews. During the investigation of the practical situation of product development, the author talked with experts in the respective domains. Especially working out the domain-specific information models showed interesting side-effects: the detailed analysis of relevant objects and relations helped also very experienced experts to gather new insights in their own work they have done since many years. This concerns the exact meaning of individual notions, certain correlations between them and even the very tasks to be performed as such. From this, it is concluded that there are still potentials to improve the processes: although engineers are commonly experts on their fields, there is still room left for raising the awareness and understanding for their own tasks and for their role within the overall process. Consequently, also the used tools and the applied methods can be optimized. The correctness of these assumptions was confirmed by the experts themselves after the investigations were finished.

Some Details on Information Modeling

In the following, some spotlights are directed on the very task of information modeling within work groups. Conclusions of further relevance will be drawn.

Bottom-Up Interviews and Workgroups with Powertrain Experts

Several activities in the automotive powertrain domain contributed to fill UMEO with feature and other EO classes and with relationships between them.

- Specific **challenges** in the powertrain domain arise by the **complexity** of mold and die tools, the continuous improvements of MDTs and the seamless integration of found features and other EOs as well as the ProSAp integration into the multi modeling approach. For example, there are many

Footnotes

[*] Relations with any number of partners, i.e., also relations with more than two partners

parts, which cannot be adapted to individual cylinder heads just by adjusting parameters: inner and outer cores and their respective details; all-in-one-casting melts two cylinder heads into a single one for cost reduction.

During a one-year period, the author interviewed designers of mold and die tools for cylinder heads and engine designers with the focus on identifying appropriate new EO types and relations between engine design and M&D tooling. During these **field studies**, **EO constellations** (EOCs) for the cylinder head design and tooling were identified. Each of the given EOCs comprises several EO classes and an EOR. Examples are *Ejector, Hole with boss, Core hole for inner core*, and *Bearing block for outer cores*. The principal usage of EO constellations is described in the section Chapter 12. The practical relevance of the found EOCs in terms of benefits on usage ranges from “very high” to “medium”[*].

To be sure to take into account as many relevant geometric elements of a cylinder head as possible during the search for new feature types, a second EO set has been identified, partly influencing the outer contours of a cylinder head. This could have been interesting for an automatic derivation of outer cores of the casting die, which turned out to be too complicated, however. Therefore, pertinent EOs are proposed to be used as *Link-Only EOs* (see below). In addition to this, EO types have been identified that require a working informational integration between process steps in order to bring positive effects; they provide downstream applications with upstream information, instead of producing straight-forward time savings for the designers. Amongst them are the following:

~ **Fixing holes** describe the positions in which a cylinder head, for example, is fixed for machining and are relevant for planning the machining resources.

~ Certain kinds of **reference planes** influence the inspection planning. They can partly be integrated into other features such as spark plug holes.

Conclusions. As a conclusion from the difficulties[†] to define new types of features (EOs) from/for certain geometric areas on the cylinder heads (basically outer contour elements), and taking account of the tight dependencies between cylinder head roughpart contours and the M&D tools, it is suggested to introduce ***Link-Only EOs***: such *ad-hoc featurization* of EOs whose geometry is represented by solids that are hard to formalize or not yet formalized, leads to advantages by producing independent objects that are as such attachable with engineering meaning, functional and other information. They can be **anchor points** for relations integrating the process steps by allowing bi-directional exchange of information.

☆ A cylinder head’s outer contour portions, for example, can be referred to in the tooling step. They can be applied by the M&D tool designer or the engine designer, by first modeling the non-feature-based geometry and then, in the second step, by manually selecting the geometry and declaring it to be an instance of a predefined link-only feature class (feature identification). Examples can be inner and outer cores, profile ledges, combustion chamber contours, and intake and exhaust ports.

Covering logically dependent EOs and their inter-relations, ***EO constellations*** have the potential to serve as **building blocks** and to represent multi-directional **associativity** (see the prototype *Ejector of Cylinder Head Mold and Die Tools* discussed on page 162). Figure 41 gives an impression of the relatively large number of relations between EOs in multiple CAD models, comprised within the EO constellation *Hole with Boss*. The same holds true for the EOC *Ejector*. The experts expected benefits from a remarkable number of EO constellations in the domains regarded.

Footnotes

[*] According to an assessment by domain experts

[†] Also in cases where it is not too difficult it often does not pay off to describe a geometry that is unique over all cylinder head series.

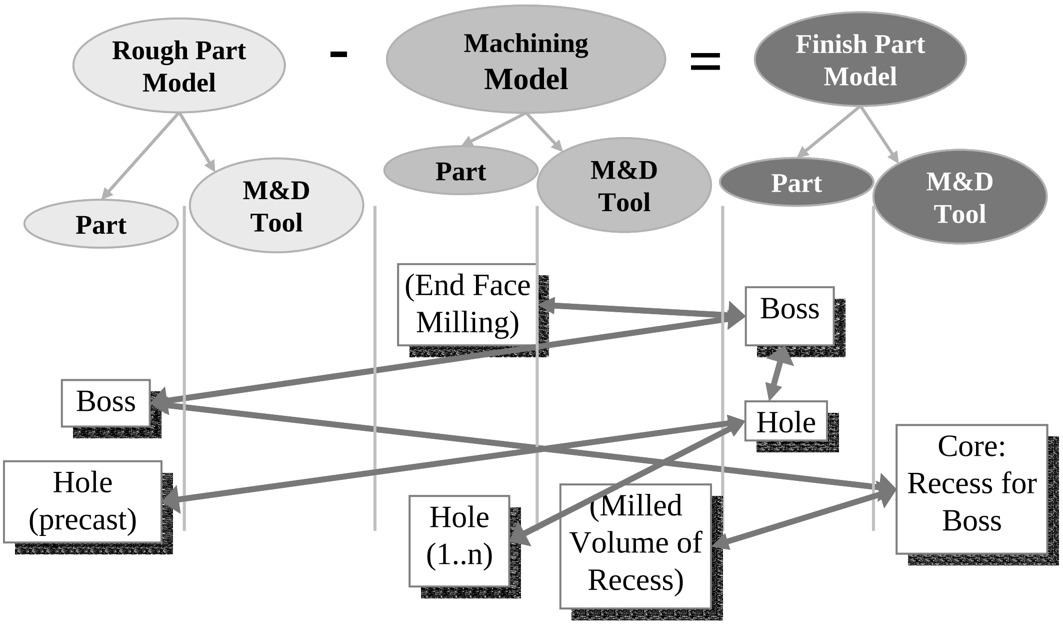


Figure 41: Hole with Boss: Relationships between EOs

Bottom-Up Interviews with Body-in-White Experts

During an OEM-internal research project, **design features** for body and white have been identified for use in car development inside CATIA^{TM} V5. Accordingly, the implementation variants *PowerCopy* and *User-Feature*[*] have been chosen in parallel. While the first allow more insights to the user in the test phases, the latter avoid those consciously during the productive usage of the feature by means of encapsulation. The features were arranged in a flat list without any hierarchy and were offered to the designer through CATIA *catalogs* (a kind of directory-based organization).

The resulting list of features includes *holes* (e.g., *free-form*, *square*, *slot*), *beads*, *flanges*, and others. Their application by the designers produces benefits through time-savings. In order to make them an information source for the downstream process of **inspection**[†], such features that are of relevance from the inspection point of view have been investigated. This investigation produced a similar list of features/EOs. The synchronization of both lists was the base for the first partly integrated CAD/CAQ process at this OEM (see also section Part II – 3.1.2 *Quality Assurance for Body-in-White Parts* for a description). However, the investigations also revealed that for a sufficient information support of inspection, more information is needed than the design-specific geometry and functional specifications: inspection tasks, tolerances, measuring elements.

While going through the second phase of integrating design and inspection, interviews with inspection specialists have been held, which aimed at the illumination of the complete process of **inspection planning and programming**. As a consequence, a much more complex and complete information model evolved. The most-current version is partly included in the I++ information model. This model also covers the administrational aspects that inspection tasks are involved in. Also the field of analyzing and presenting inspection results had considerable impact. On the upstream side, design and assembly issues were more intensively illuminated. One of the most crucial informational entities in the CAD/CAQ area has been identified to be the concept of a ***quality criterion***. A quality criterion is an engineering object relation, correlating one or more geometric objects, a tolerance

Footnotes

[*] Synonym for user-defined feature

[†] Inspection of series and prototype assembly processes

information, an assembly/production step and functional information inside a single constellation. This IE is an integrated set of information that is atomic in the sense that none of its relation partners – apart from the geometric object – can be separately considered without losing a vital part of the quality information. In other words, the related information does not make sense if one of these elements is missing: the function of a geometric object determines the tolerances that are to ensure its fulfillment by the manufactured geometric object, and tolerances are always dependent on manufacturing steps. The Quality Criterion EORM is depicted in the center of Figure 42 next to the *Inspection Strategy* EORM. Both correlate IEs from several domains.

Conclusions. Relations in general, and especially **n-ary relations**, play a central role in the inspection information model, as they correlate IEs belonging together inseparably. N-ary EORs cannot be replaced by binary ones without losing information or transparency and naturality of the information model.

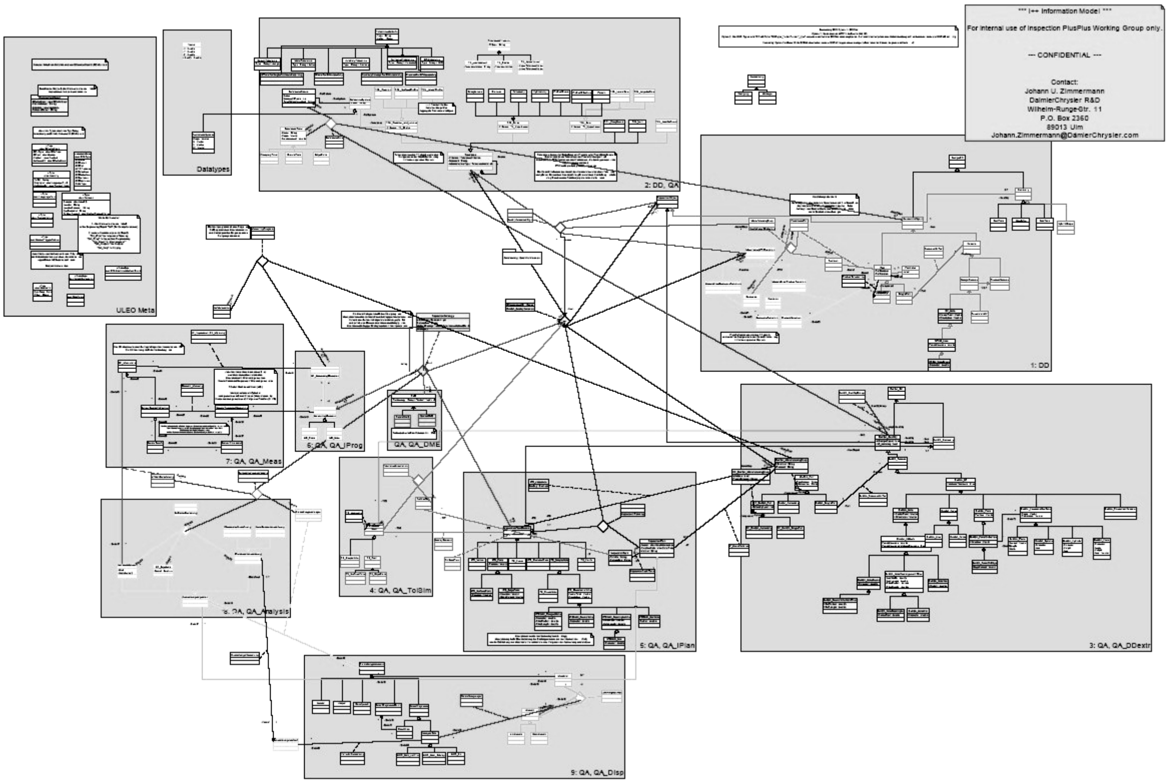


Figure 42: Large-Scale View of the I++ Information Model (UML); grey boxes depict domains

Inspection-plusplus Workgroup

The *Inspection-plusplus* workgroup has already been introduced in section *Inspection-plusplus* on page 28. In this workgroup, experiences of experts of several automotive manufacturers are compared and integrated. Thus, the information model (see *[Kolb, 2001]* for a first release), became universally applicable without losing its specific character (see *[Zimmermann, 2004]*). For example, a user-defined tolerance has been introduced and quality criteria became recursively aggregatable.

☞ The I++ information model represents a **semantic kernel** within the IIM.

A large-scale view of the current I++ information model is depicted in Figure 42 in order to give an impression of the complexity. The development of the information model has been carried out **top-down**. First, one process step after the other was considered in its global context, and after that every

EO class and relation was further detailed and specialized[*] – but this modeling process happened also **bottom-up**: use-cases from the quality assurance domain were described delivering important detail information, which allowed the knowledge engineers to assess the practical applicability of the information model in terms of completeness and expressive power.

The availability of elaborated and n-ary **relations** was used even more intensively for representing quality assurance information. This allowed a clear structure to be maintained while minimizing the need for explanation and for constraints, associating two or more relationships. In fact, almost no constraints were needed. An interesting aspect of the information model comprises the various EORs representing **strategies** for inspection programming and for analyzing and visualizing of the results. These strategies can partly be automated in that they store the knowledge as interpretable code within their methods.

Conclusions. The conclusion of the preceding section is supported by the I++ efforts: **sophisticated, n-ary EO relations** are **vital** in the inspection information model: methods are used within relations for various kinds of inspection strategies, n-arity welds together what belongs together.

Solutions to Practical Modeling Tasks

Due to a lack of space in this thesis, several issues, having occurred during this work, will not be discussed in detail. Amongst them are the modeling of inspection strategies, the modeling of product variants using EORs, and *Aggregated Objects* and *Redundant Component Relations*, and the task of finding EOR types (EOR analysis).

Conclusions drawn from these issues are the following: Some of the discussed domains show rather complicated relationships between entities, most naturally being represented via **n-ary relations** as proposed in the ULEO approach. This includes also **relations between relations** and other EOx. The used examples showed that relations allow information to be organized redundancy-free and suitable for downstream processes (quality criteria are specifically geared for inspection) without influencing the very EO classes.

To sum up: Dealing with practical information modeling situations shows that relationships play a vital role in many domains and that the provision of sophisticated relations by the IT systems eases the modeling and leads to more natural models. Frequently, EORMs are central informational entities on which development processes base their respective work. While EORMs are part of the company's background knowledge, e.g., by representing methodological knowledge, EOR instances do the detailed tracing of relationships between contents of various product models.

User's View of the Prototype

☞ The functionality of the respective software implementation is far too complex to be described in detail within this thesis. The same holds true for the user interface. Therefore, the following sub-sections will only give some impressions on the tools from the users' perspective.

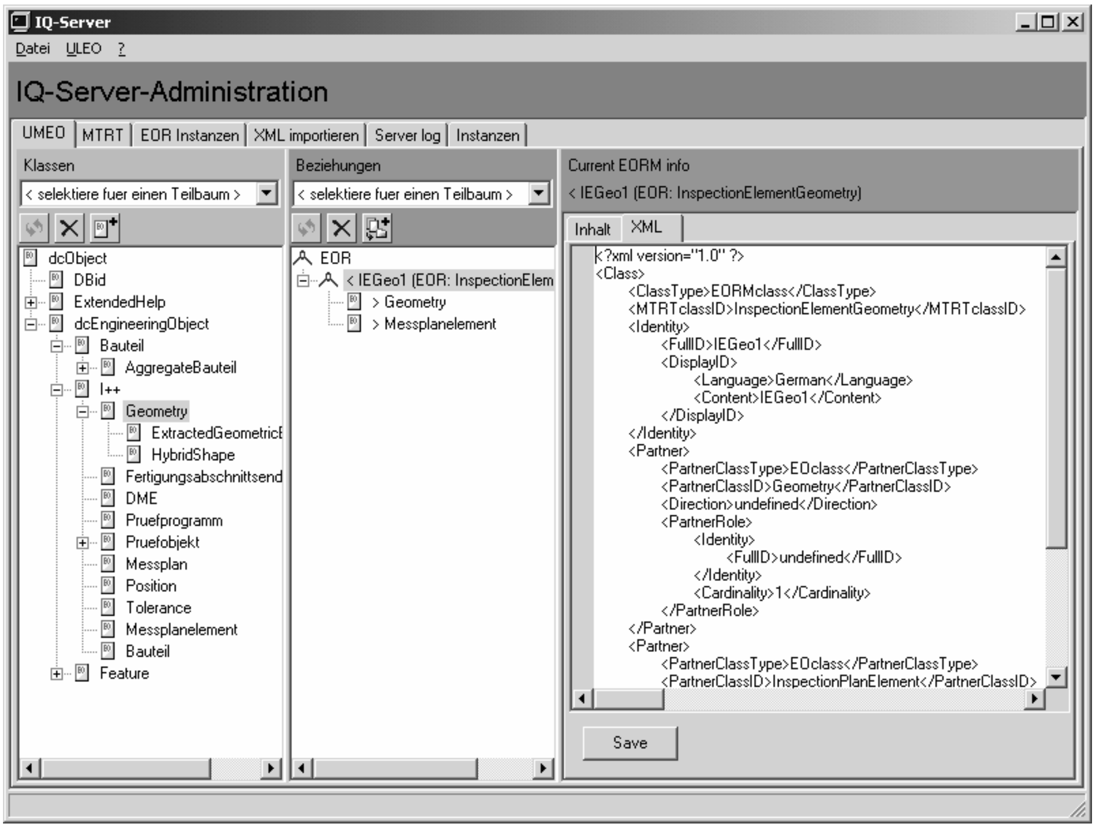
ULEO Server and Administration Client

Users of the ULEO server are no product engineers but IT system administrators. The administration client takes over the functionality of a user interface of the ULEO server. Figure 43 gives an impression of the *ULEO admin*'s human-machine interface (HMI).

Footnotes

[*] Utilizing inheritance relations

The ULEO admin software application prompts the user for login and displays a three-tired panel, fore example, displaying UMEO contents: the left column for the EO class taxonomy, the center column for EORMs, the right column for the contents of the EO or EORM currently selected within the left or center column. The right column can be displayed in a tree-like view or as editable XML text. UMEO EO classes can be browsed, while the center column is able to display any n-ary relation. Other tabs are for editing the MTRT, EOx Instances, XML import/export and logging. The current implementation has not yet reached its full functionality.



```
<?xml version="1.0" ?>
<Class>
    <ClassType>EORMclass</ClassType>
    <MTRTclassID>InspectionElementGeometry</MTRTclassID>
    <Identity>
        <FullID>IEGeo1</FullID>
        <DisplayID>
            <Language>German</Language>
            <Content>IEGeo1</Content>
        </DisplayID>
    </Identity>
    <Partner>
        <PartnerClassType>EOclass</PartnerClassType>
        <PartnerClassID>Geometry</PartnerClassID>
        <Direction>undefined</Direction>
        <PartnerRole>
            <Identity>
                <FullID>undefined</FullID>
            </Identity>
            <Cardinality>1</Cardinality>
        </PartnerRole>
    </Partner>
    <Partner>
        <PartnerClassType>EOclass</PartnerClassType>
        <PartnerClassID>InspectionPlanElement</PartnerClassID>
```

Figure 43: ULEO admin: EORM View of UMEO – XML Text Editor Tab

Inspection Planning Tool MPE

After the finish-part has been released, and tolerances have been attached by the designer and/or the tolerance work group (using CATIA^{TM} V5 workbenches *GSD* and *FTA*), the inspection planner can begin work using the MPE tool (Figure 44 portrays a typical situation).

The results of inspection planners interactive work will be visualized by the MPE and stored in the EO(R) instance database also managed by the ULEO server (part of the ULEO database). During the inspection planning process, also aspects of inspection programming – the subsequent step in product development – can optionally be performed: the engineer may online access inspection strategies (see also section *Inspection Strategy Assistant*) stored within UMEO and chose the most promising one for each inspection plan element. If doing so, the inspection programming can be performed largely automated by the inspection programming tool, and controlled by the runtime interpretation of the inspection strategies selected before within the MPE. As the MPE uses the ULEO server also to store EOR instances, the consequences of changes in the CAD or tolerance information can be traced down along the entire inspection planning and programming process. The inspection results are currently not yet integrated into the global information space but are intended to be so.

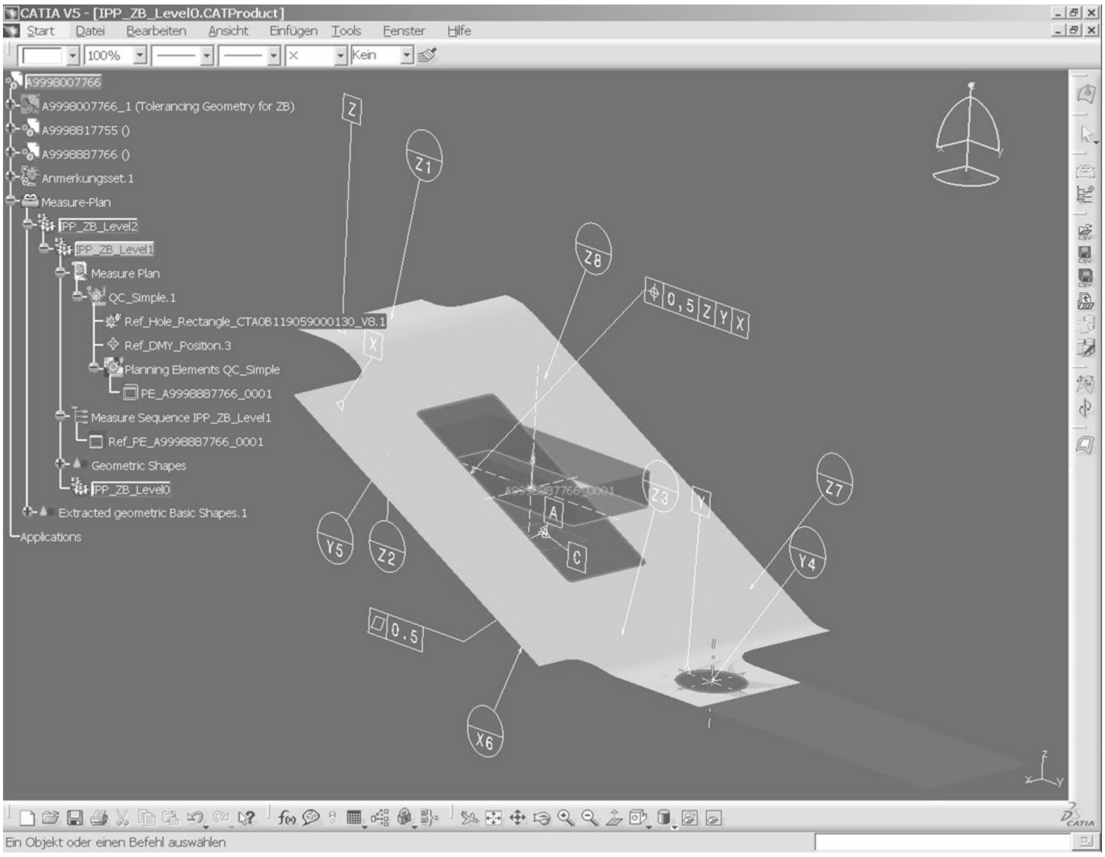


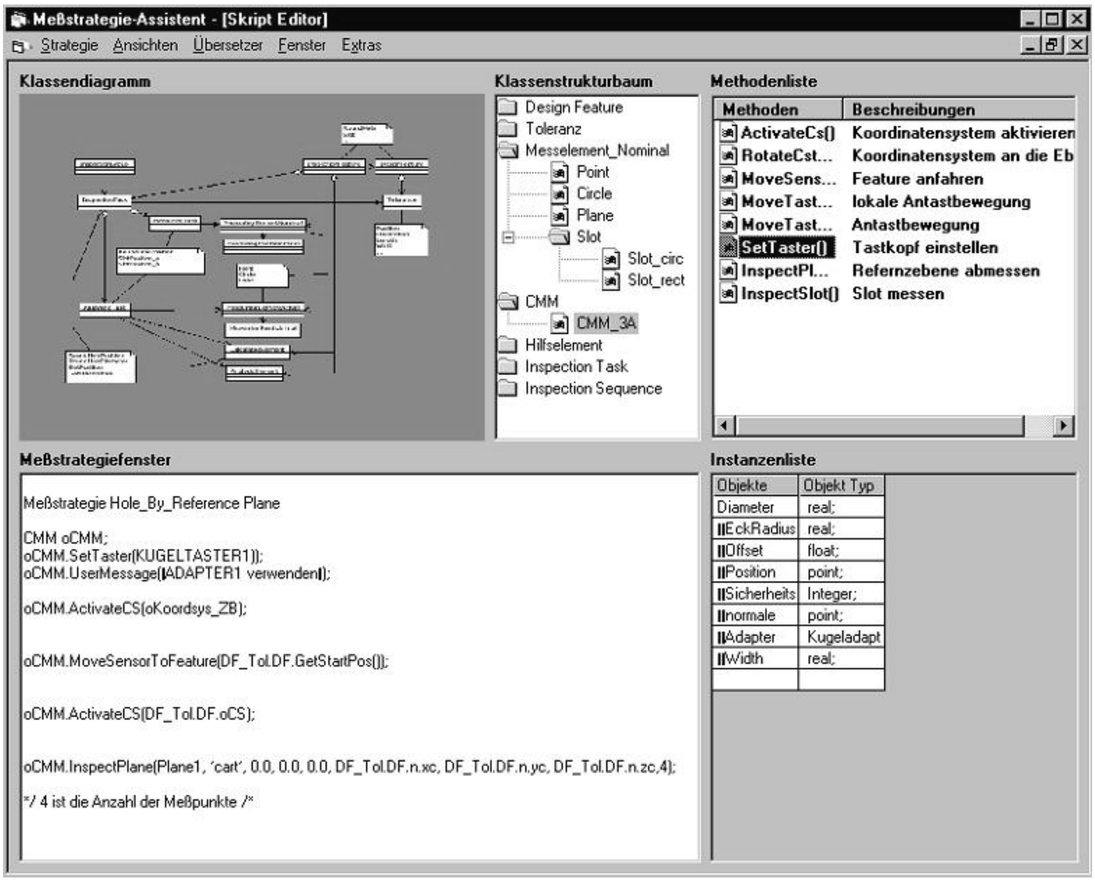
Figure 44: Typical Impression of the MPE HMI

Inspection Strategy Assistant ISA

The ISA will support mouse- and icon-based manipulation of inspection strategies and, in parallel, produce ISRL code (see Figure 45). Direct manipulation of the ISRL code will also be possible. ISRL is object-oriented and EOx can be used directly within the code. All EO classes and EORMs that are used within the code, will be specified as relation partners of the *inspection strategy* EORM in order to denote inputs and outputs quickly and to protocol involvement of such IEs within the strategies. The tool’s final version will have to consider the requirements coming from the sophisticated user interface including dynamic visualization of UMEO contents combined with a drag and drop functionality into the ISRL text window.

Discussion and Possible Enhancements

The prototypical and productive software applications described implement the full-fledged ULEO concepts. Although centralized automation using a ULEO server has not been implemented yet, information support and decentralized automation have been **implemented** and turned out to be implementable with a **limited effort** of about one and a half man-years for the server and some additional 8 man-months for the first release of the MPE software.



Methoden	Beschreibungen
ActivateCs()	Koordinatensystem aktivieren
RotateCst...	Koordinatensystem an die Eb
MoveSens...	Feature anfahren
MoveTast...	lokale Antastbewegung
MoveTast...	Antastbewegung
SetTaster()	Tastkopf einstellen
InspectPl...	Refernzebene abmessen
InspectSlot()	Slot messen

```
Meßstrategie Hole_By_Reference Plane

CMM oCMM;
oCMM.SetTaster(KUGELTASTER1));
oCMM.UserMessage(▌ADAPTER1 verwenden▌);

oCMM.ActivateCS(oKoordsys_ZB);


oCMM.MoveSensorToFeature(DF_Tol.DF.GetStartPos());


oCMM.ActivateCS(DF_Tol.DF.oCS);


oCMM.InspectPlane(Plane1, 'cart', 0.0, 0.0, 0.0, DF_Tol.DF.n.xc, DF_Tol.DF.n.yc, DF_Tol.DF.n.zc,4);

*/ 4 ist die Anzahl der Meßpunkte /*
```

Objekte	Objekt Typ
Diameter	real;
▌EckRadius	real;
▌Offset	float;
▌Position	point;
▌Sicherheits	Integer;
▌normale	point;
▌Adapter	Kugeladapt
▌Width	real;

Figure 45: ISA Illustrator

The **Python** programming language, which was used for the implementation of the ULEO server and ULEO admin client, allowed fast and effective coding and parallel testing utilizing the Python interpreter. Also, the existence of many standard libraries was helpful to create user interfaces, for example for the administrative client. Even the CORBA communication, often thought of as a rather complex task, could be done quite quickly this way.

The **GIS information structure** proved to be sufficiently expressive for all the situations encountered and led to intuitive and adequate results. It opened up the way to natural information models. This structure, together with the ULEO database, turned out to be open to any extension in terms of new sorts of information and was therefore a useful information carrier. Navigational access to them proved to be feasible, robust and efficient. For instance, all the available inspection strategies for a given quality criterion could be retrieved efficiently, without undesired results from UMEO.

How to Achieve Integrated Inspection

An integrated CAD/CAQ process chain requires adding further ULEO clients for inspection programming and inspection analysis. This will be simplified by the achievements of the I++ work group as there is a common semantic kernel in the IIM.

Holos NT^{TM} and *Calypso*^{TM} are able to access measuring equipment online. The measuring results are stored in the respective proprietary databases. A major further step in integration would be to additionally include these databases into the GIS. The adoption of the ULEO GIS information structure by the software vendors implies tracing relations from inspection planning instances to the respective results.

✍ If these databases are integrated into the GIS, it does not seem necessary to internally change the measuring results databases to a ULEO-formatted storage as used within the ULEO database; the integration add-on will perform a translation of the structures. However, the information models underlying the database contents must be logically compatible with the semantic I++ kernel of the GIS, if a maximum integration is to be achieved.

☞ “**GIS DMA**”: Measuring result databases are heavy-duty databases, in that they have to swallow huge amounts of data in very short times. Therefore, a generally applicable approach is suggested for integrating such applications into the GIS: GIS integration works on a sophisticated level if, for instance, the complexity and semantic clearness of the shared information is considered. Such sophisticated integration cannot be as efficient as more specialized and restricted ones. Hence, in cases where runtime efficiency of the GIS interface does not satisfy the practical needs, it may be worthwhile to integrate a **twin-coupling**. In the case of the measuring result databases this would mean integrating them into the GIS and leaving their interface to the measuring machines unchanged. Thus, the time-critical information flow is still assured – comparable to computers’ *direct memory access (DMA)* technology – while the GIS-relevant information (whose access times are typically less critical) is still available.

Also the subsequent step of analyzing and presenting inspection results has currently not yet been practically integrated. As there are proprietary software solutions, efforts will have to be made in this field also to convince those responsible on the client and vendor sides.

Regarding the vertical dimension of the inspection scenario’s application range, plans are in line to extend it to powertrain inspection. The following section focuses on some pertinent issues.

Powertrain Aspects

On the I++ side, a key issue is how to assure usability of the information model for all practical application fields of quality assurance within the automotive industry. Deviations will occur in the functionality of software applications, as the instantiation of the information model contents differs partly.

✍ These differences result basically from the fact that the geometry of powertrain parts changes during manufacturing (machining) only in the machined areas, whereas sheet metal parts may change their overall dimensions due to their inherent flexibility. As a result, powertrain parts are inspected after each machining sequence in the machined areas only, while sheet metal parts are potentially re-inspected using different tolerances.

Inspection **strategies** are **structured** differently for powertrain and sheet metal parts, as the technological variance of inspection equipment varies. Thus, the border lines between technology-dependent and -independent (sub-)strategies are on a different level, which means that the technology-independent powertrain strategies also contain more assumptions on technology than they do in sheet metal.

Although the information model is shared on the level specified by I++, the **feature types** used differ according to the respective application domains. Powertrain features are basically different from sheet metal features, even if certain ones could be shared. Powertrain features are commonly prismatic and can be decomposed more easily; simple holes, for example, may be used for generating stepped holes, and the **inspection strategy** for a compound hole **may be derived** from the simple holes’ strategies.

A motivation for **sharing features** can be the wish to reduce the variety of products, manufacturing tools, software tools, and inspection methods. From this list, only the software tools remain as candidates for possible simplifications, as, today, the practices of developing and producing sheet metal parts significantly differ from those employed for powertrain parts. The sharing of feature types offered by software tools is completely feasible in inspection, as pointed out in the preceding paragraphs. Use of design features, however, seems to hardly pay off as long as incompatible CAD modules of the same software vendor exist for handling sheet metal and prismatic parts.

13.3.2 Detail Design

The prototype covering this scenario is currently being implemented and shall serve as a basis for discussions with automotive designers. The implementation mainly concerns the extension of CATIA^{TM} by a ULEO client/server functionality and additional services inside the existing ULEO server described in the preceding section.

Scenario and Scope of the Prototypes

Scenario motivation – general. This scenario demonstrates partial automation of design tasks in that instances of engineering objects and respective EO relations (EO constellations) are created by the system. This is the sophisticated version of a functionality rudimentarily shown in the above-described first prototypes. It also shows how feature- and other EO types can be centrally managed and provided to the CAx systems.

Scenario motivation – what is new? Users navigate on up-to-date trees of EO classes; manual roll-out (delivery and installation) of new feature types becomes obsolete*. Intra- and inter-application automation is controlled by the ULEO server.

Description of the prototypes' functionality. A ULEO ProSAp (CATIA^{TM} V5, *part design* workbench) uses and offers GIS services in order to access certain parts of UMEO, thus offering to the engineer up-to-date feature trees for instantiation inside the finish-part model. Automation EORs within UMEO are displayed and – after a user command – executed by the ULEO server in order to have the CAD system perform automatic instantiations of EO(R) classes, triggered by events.

Technical Details

CATIA is extended by a C++ ULEO add-on using the CAA development environment for *Microsoft Visual C++*. An additional toolbar and command buttons let the user access the new functionality. On pressing the *Instantiate EO* button, the add-on reads EO classes and EORMs from UMEO via GIS services. If *Automation* has been chosen by the user, the ULEO server calls services of the CATIA add-on (now acting as a GIS server) to create instances of EO classes, but also instantiates EORs inside the EOR instance database to correlate them. For this purpose, the ULEO server described in the quality assurance scenario has to be extended by the event handling functionality described in Chapter 11.

Central handling and roll-out of EO types is achieved by storing the CAD-specific definitions (in case of CATIA called *user-defined features (UDFs)* or *User-features* and stored within individual CATIA models) on a central directory managed by the ULEO server, and by referring to each of these files from within the according EO class within UMEO. If a user decides to instantiate a certain UDF, the CATIA add-on retrieves the location of the respective CAD-specific file from the EO class description and requests this file from the ULEO server by calling a specific GIS file service. Subsequently, CATIA possesses all information necessary to instantiate the new feature type.

User's View of the Prototypes

The design engineer is working on a part of the automobile by instantiating features and other EOs into a CATIA V5 model. Each instantiation is initiated by pressing an I*nstantiate EO* button, followed by a dialog that displays a *Windows Explorer*^{TM}-like navigation tree.

Footnotes

* Another intermediate approach using a online web portal is shown in *[Schwarz et al., 2002]*

The designer selects the desired EO class while getting more detailed information about it in the right part of the dialog after selecting them in the tree on the left-hand side. This information includes attributes, but also optional information about automation services offered for the currently selected EO type. Further automation information can be accessed by pressing a button that is activated if pertinent information is available. Basically, automation information is represented by EORMs of types known to the ULEO add-on. It is displayed if one of the EO classes involved in a given EORM is selected by the user. Thus, the same EOC can be accessed by starting off from several EO classes. The idea behind this is to offer **automation information as add-on** to the central building blocks of the engineer, instead of letting him/her navigate through a set of automation functions. The user can directly chose *Instantiate EO* or press the respective *Automation* button in order to proceed through the subsequent steps. If *Automation* is selected, the corresponding EORM is executed by the ULEO server and the user can view the corresponding steps. Each step optionally[*] prompts the user for confirmation. The result are one or more instances of EO classes and EORMs created within one or more CAx models.

Discussion and Possible Enhancements

This implementation of the EOC does not share the restrictions of the first prototypes for EO constellation linking (see sections 13.2.1 and 13.2.2 and following): for example, the descriptions of EO classes and EO constellations are read from UMEO and MTRT online and can therefore be manipulated without changing program code. The inherent scalability of automation together with the avoidance of manual EO class roll-outs show interesting functionality for initial practical applications and are thus a basis for discussions with end users and the IT personnel responsible.

13.3.3 Brief Glance at Partially Implemented Application Scenarios

The universal prototype has been supplemented by add-ons to CATIA^{TM} V5 workbenches in order to show the principle benefits within other scenarios relevant in the near future. Accordingly, they have been implemented to the extent necessary for a clear demonstration.

Extended Help Scenario and Prototype. This scenario works with a CATIA V5 C++ extension offering context-specific help information to a designer. This help information relates to an object (EOI) selected[†] within the CAD model. Help information is determined based on the object’s type, which is corresponding to an EO class within UMEO. The add-on retrieves an EORM of a given type this EO class is involved in. One path of this relation leads to a “help” EO class within UMEO, within which one certain attribute contains a URL that refers to an http page provided by a knowledge processing tool’s Web Service. The URL creates and provides the information desired (specific to the EO class of the selected CAD feature). The CATIA add-on retrieves the URL and starts a web browser displaying these feature-type-specific web pages generated by the knowledge-processing tool.

The objective is to demonstrate the context-sensitive provision of sophisticated knowledge about CAD model contents. This provision is directly triggered from within the CAD tool, so the knowledge directly supports the engineer’s practical work. The indirection within UMEO demonstrates the flexibility and structurability of background information, as different types of help-EORMs could lead

Footnotes

[*] According to the configuration of the ULEO add-on in CATIA and/or the meta-information inside the EORM

[†] In the sense of “marked” object

to different kinds of additional information. Alternatively, external referencing[*] could have been used to access background knowledge represented in terms of EO classes and EORMs.

Twin-CATIA Prototype

This scenario demonstrates an information flow between two installation of the CAD system CATIA^{TM} (two applications running the same add-on), one of which logs on at the ULEO server using the domain name *detail design* and the other logs on using *machining*. In both applications CAD models are loaded. The detail designer wants to get information on the machining of a selected design feature. After starting the respective command by pressing a button within the CAD system, the user selects[†] a design feature instance. Subsequently, the add-on retrieves an EORI of the type *is_machined_as* leading from the design feature instance to the machining feature instances managed by the second CATIA installation. The respective machining feature instances are retrieved through the GIS services offered by the ULEO server, while specifying their EOx addresses obtained from interpreting the EORI contents. These EOx addresses contain, amongst others, the domain name *machining* through which the <u>second</u> CATIA installation can be identified by the ULEO server. The IDs and attributes of the machining feature instance retrieved are visualized to the detail designer.

This implementation practically illustrates the deployment of the ULEO addressing schema[‡] in a Design-for-X scenario. In a productive environment the add-on could, for instance, also retrieve cost information and experiences on manufacturability, starting off from the machining feature's EO type. Thus, the designer would get an idea about costs and possible problems of the design solution chosen. If this information depended on further elements within the current detail design CAD model, the application serving as information source, could perform queries to the design CAD application (via the ULEO server).

13.4 Experimental Prototype for an Application-spanning Automation Scenario

The conception and implementation of this prototype served to assess an alternative implementation architecture for some of the ULEO concepts. This section sketches the scenario and the prototype. Refer to diploma thesis of Ekkehard Steiss for details (see [Steiss, 2003]).

Scenario and Scope of the Prototype

Generative EO relations (GEORs) represent small portions of workflow elements (see section Part IV – 9.4.1). The *MicroFlow / Process* technology based on IBM's *Websphere Application*^{TM} middleware was investigated as a commercially available candidate to realize application-spanning and flexible automation based on the execution of GEORMs. In this context, three scenarios were investigated, implemented and assessed, beginning rather simple and stepwise implementing more complex functionality.

Footnotes

[*] See the section *One-Entrance Principle and External Referencing* on page 110.

[†] In the sense of "marks"

[‡] See the section *Identification and Addressing Schema* on page 122.

Each scenario was used to assess the fulfillment of specific requirements placed on the software:

- Scenario 1: Exchange of EO instances between ProSAps
- Scenario 2: EO instantiation within a single ProSAp
- Scenario 3: EO instantiation within several ProSAps and controlled by the WAS[*]

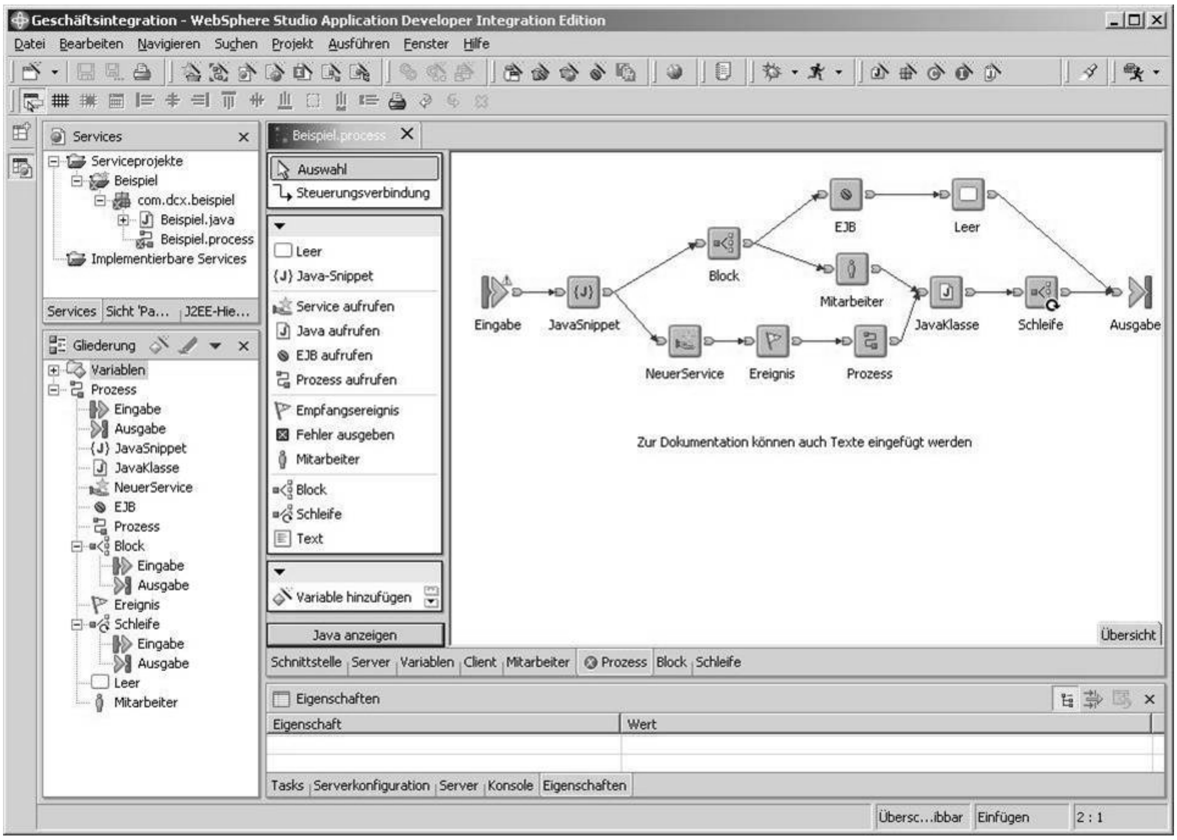


Figure 46: IBM Process Editor [Steiss, 2003]

Technical Details

The integrated development environment *Websphere Studio Application Developer Integration Edition, Version 4, (WSADIE4)* was applied for developing Microflows. The next version, *WSADIE5*, offers so-called *Processes* instead of Microflows (see Figure 46). As *Processes* turned out to be more flexible than Microflows, all scenarios have been implemented using WSADIE5. *Processes* are designed to be executed on a *Websphere Application Server, (WAS)*. A simplified UMEO was stored in a Microsoft Access^{TM} database. ProSAps were simulated by dedicated services on the WAS. Information retrieval was performed purely set-oriented.

User's View of the Prototype

Do to the IT-technical nature of the investigations, the users' view was not considered. For the same reasons and due to a lack of time, ProSAps were just simulated.

Footnotes

[*] Websphere Application Server^{TM}

Discussion and Possible Enhancements

Eight detailed requirements assumed to be basic for implementing ULEO concepts have been assessed and evaluated within the three scenarios (for more details, please refer to *[Steiss, 2003]*). The central result of these investigations and implementations is that a commercial middleware solution – represented here by IBM's Websphere Application Server – can, in principle, be used for implementation of a CAx-application-spanning automation. However, the integration of ProSAps within a global information space based on dynamic information contents in IIM/UMEO, as has been described in this work and stated to be vital for the future product development process chain, cannot directly be achieved with the standard WAS methods. For this purpose, special software implementations are necessary. Also, the available pertinent IT development environment is unnecessarily complex and large, since it targets much larger domains and offers several services superfluous from the ULEO perspective. As a result, the scalability of the solution decreases significantly, which must be judged a serious drawback. Finally, the sorts of ProSAps have to be known to the middleware in advance, which hinders ProSAps' dynamic logging on and off. However, as no principle fences could be identified, these issues will be subject to further investigations to be performed with research fellows specializing in middleware solutions.

13.5 ULEO Contributions to Research Project Scenarios

This section outlines some further projects utilizing ULEO concepts. It primarily aims at examining other application fields that have not yet been worked out or implemented to the degree of detail as the ones described above.

13.5.1 Research Project *3D Workbench*

This government-funded research project is led by the Institute *RPK* of the University of Karlsruhe and brings together university and private research and small IT companies[*]. It targets the development of a component-based software development platform (middleware) for industrial CAD/CAM/CAE applications, based on open source standards. Commercial CAx tools (currently CATIA^{TM} V5 and Pro/Engineer^{TM}) are supported with OMG's CADservices adapters (see *[OMG CADservices, 2003]*) allowing any software application to access and manipulate CAD data on a currently rather basic level – the OMG works on the enhancement of the CADservices, however. A so-called *product configuration* application uses these services to activate or generate the geometric representation of pre-defined product configurations. Web browser- and VR[†]-based viewers allow engineers to inspect this geometry through the same information paths. The information flow between the CAx systems and the viewers is improved by supplementing the CADservices interface by ULEO-XML-formatted feature instance information provided by the CAx system. The second informational "add-on" consists of ULEO XML files for describing EO classes and viewing meta-information such as requirement types to be visualized in the RPK's VR viewer. Due to the limited project resources, these basic solutions have been chosen to be implemented first, while a ULEO GIS client functionality will be implemented within the VR viewer application subsequently. In this constellation, the 3D Workbench architecture harmoniously complies with the ULEO architecture proposed in this thesis, while CADservices can be considered as part of the ULEO IPCI, and the viewers are ULEO GIS clients and optionally servers. This project's application scenario emphasized ULEO's applicability in

Footnotes

[*] See the URL *http://projects.opencascade.org/3dwb/*

[†] Virtual reality

pushing non-proprietary information flow and makes use of the common information model UMEO, where background information is stored and accessed by the individual applications.

13.5.2 Research Project *ISoMEEr*

This OEM-internal research project aims at the development of new methods and tools supporting three disciplines from an integrated point of view in order to support Design-for-X. *ISoMEEr* stands for *Integration Software, Mechanics, Electrics/Electronics*.

It is planned to utilize the ULEO concepts and to apply the ULEO server developed for above-mentioned projects to integrate the relevant commercial CAx tools with tools for functional modeling and knowledge processing. There are no prototypical implementations available yet.

Especially interesting from this research's point of view is the representation and management of mechatronic constellations, correlating – being a special case of EO constellations – EO classes (and instances) from the above-mentioned domains and thus providing integrated building blocks for the engineers' inter-disciplinary work. EOR instances provide the dense network of relations necessary to track individual designs' internal correlations.

13.5.3 Research Project *Integrated Industrial Image Processing*

This OEM-internal project tackles the realization of automated product assembly checks by visual recognition of engineering object instances. For this purpose, starting off from the finish part information in CAD files, image processing is performed by several software applications, finally resulting in a neural object recognition system. The project's final vision integrates the CAx system and the object recognition system into the global information space and stores strategies for image processing and object recognition within EORMs in UMEO: following the ULEO concepts, these downstream processes are provided with dedicated EO sets correlated by the mentioned EORMs. Partial automation of these domains' tasks is targeted.

Especially interesting from this research's point of view is the integration of a ProSAp processing sub-symbolic IEs, into the object-oriented GIS.

13.5.4 Research Project *FAD*

FAD is the acronym for *Feature- and Knowledge-Based Assembly Design*. Special emphasis is placed on the integrated modeling and processing of product functions and product templates, which are pre-defined building blocks for product detail design comprising a rather large number of EOs, e.g., for design of complete assemblies and sub-assemblies of a car body-in-white.

In this OEM-internal project, ULEO is to take over the role of a CAx integration solution. For example, product functions are to be integrated into the information processing of the *MPE* ULEO client described in the section 13.3.1, thus informing inspection planners to about the functions that certain *quality criteria* are meant to assure. To achieve this, product templates are equipped with sophisticated function models (see also *[Leemhuis, 2004]* and *[Leemhuis et al., 2002]*).

Chapter 14 Software Tests

This section focuses on the software described and discussed in the section 13.3.1 Quality Assurance for Sheet Metal *and describes the tests performed to find out if the software meets the requirements. The software applications considered are the* ULEO server, *its administration client* ULEO admin, *and the inspection planning software* MPE. *A validation of the ULEO approach as such is given in the conclusion part of this thesis.*

☞ Details of the tests, down to individual **test criteria**, will not be given here, primarily due to space constraints. Instead, the major results will be explained according to several key criteria.

The criteria according to which the software applications were to be assessed were applied within the **test methods** set out in the following. The ULEO server was tested by evaluating its effects as visible through the client applications and partly by logging outputs on the screen and into log files. In some cases, special routines have been built in to provide the desired information. In order to estimate the robustness, test clients simulating real ProSAps have been coded and started cluster-wise to try to put the ULEO server under pressure. The MicroSoft Access^{TM} database has been inspected manually using Microsoft's standard application. A small connection test client has been developed in order to observe the connectivity from various locations within the company's TCP/IP intranet (Germany-wide) on both sides of several firewalls.

The clients ULEO admin and MPE have been tested by performing all the specified actions both correctly and intentionally incorrectly. Temporal behavior was checked using measuring routines within the server and the clients. Storage and retrieval of data was checked within the ULEO database and on the GUI.

Carrying out the tests. The tests have been performed by end users and/or software engineers at different times and locations.

Summary of the test results. As most failures and faults had been cleared or removed prior to testing, the criteria regarding software **functionality** were almost fully met by the applications: the GUIs have been judged as "good" by the end users, and **reliability** has reached the required state so that the application can be called **robust**.

The **efficiency** of the applications involved and of the ULEO IPCI used, turned out to be sufficient for the above-mentioned purposes within quality assurance, although the interpretation of XML at runtime is relatively time consuming. One of the next tasks to perform is to check the applicability to online mass data processing, which occurs during storage of actual inspection results and means an additional tightening of speed requirements. Please refer also to the keyword "GIS DMA" in the description of the prototype. Should shortcomings regarding efficiency occur, several options are available for optimizing the processing speed: Python code can partly be replaced by C++ code, caching strategies can be further optimized, the ULEO IPCI can be extended by more complex services[*] reducing the need for many individual service requests and delivering larger XML blocks, each comprising numerous informational entities. Finally, database system-specific functionality can be utilized. Thus far, no case of timed-out http communication has been detected (see section 11.2.2).

Footnotes

[*] Offering, at the same time, more comfort to the clients

Part VI – Conclusions and Recommendations

This part of this thesis offers a detailed reflection on and assessment of the appropriate solutions proposed for the goals. After an update of the state of the art as surveyed in earlier chapters, recommendations are given. Finally, the reader will be informed about the author's future work.

Chapter 15 Overview of Achievements and Restrictions of This Research

This chapter briefly recalls the course of this research, before it characterizes the major achievements and restrictions. Please refer also to section Part IV – 9.2.2 What is (not) new? *for more correlated details.*

Upon looking at the current situation of product development in the automotive industry, the author was able to identify the following **key challenges** in the related IT domain:

~ **Informational integration** of applications in product development by embedding them into a global information space GIS; thus closing the informational gaps between the applications, raising the availability of sophisticated information significantly, while also considering legacy software applications.

~ **Automation** of routine tasks to achieve more standardized processes and product components by bringing a subset of the company's know-how into daily use: this manifests in the form of partly machine-interpretable standard strategies for recurring tasks and experience handling to avoid errors.

Regarding practical conditions in the automotive industry, it has been stated that new software concepts that are scalable and able to cope with the current software world and – even more – able to integrate this imperfect world as far as possible have much better chances of being introduced.

The resulting approach has been given the name *Universal Linking of Engineering Objects (ULEO)*. To repeat the above motivation for this naming, "the method of linking logically related engineering object classes and instances is regarded as the key for both informational integration and automation. Although the approach defines more than simply linking objects, this term emphasizes the crucial importance of a sophisticated relation concept within a GIS and points to product engineering as this approach's ultimate destination domain. The term 'universal' claims that there is no limitation of applicability to certain sub-domains in the product development that ULEO is geared for."

ULEO identifies and specifies solutions for creating a GIS between applications in product development. These **solutions** comprise the following:

~ A specification of appropriate basic information types within a GIS

~ A representation formalism *ULEO XML*

~ A specification of GIS services supporting information and control flow

~ A software architecture

~ Specifications for applying ULEO (description of new processes and GIS contents)

~ Prototypical software and databases

Restrictions concerning the goals. The ULEO approach tackles all of the stated goals, so that there are no restrictions in theory. However, not all of the theoretically suggested solutions have been implemented yet. Since the current implementation of the ULEO-based software in the quality assurance domain is also considered to be a pilot for the further application of ULEO concepts in other domains, exactly the functionality that is immediately needed has been implemented. Therefore **not implemented** inside the ULEO server are the following features: event handling for processing of application-spanning automation, distributed ULEO servers, and distributed databases. The same is true for IE views, and archiving and versioning of the ULEO DB. Furthermore, the utilization of generative EORs and of OO interfaces in relation types has not been implemented in software so far (hypotheses 3d+f). Concepts for data security and access control have so far been implemented using basic methods only.

Essentially, this research has contributed to driving the **state of the art** in the following ways:

- By investigating and analyzing parts of today's product development processes considering the real-world situation at a major automotive manufacturer.
- By deriving a catalog of requirements placed on solutions for IT for engineering that are specifically suited for tackling the major drawbacks of this domain. This catalog is not only a collection of demands but, at the same time, also a set of coarse solutions in itself. It is intended to be the best possible compromise between ideal theoretical approaches, on the one hand, and practical restrictions on the other.
- By deriving a bundle of fine-grained solutions, especially geared for meeting and detailing these requirements.
- Basically, it is this special combination of solutions that yields ULEO's overall benefit, not so much the invention of new stand-alone methods. Although such individual methods exist: for example, the application and specification of the notion of a global information space including the definition of appropriate informational base types, addressing, and context schemas, all flowing into an optimized representation formalism, as well as the GIS services and the global architecture. Also included are, for instance, the revaluation of the significance of relations and the introduction of GEORs. The major contribution, however, is the adaptation and combination of existing methods to achieve solutions that are custom-made to the demands of product development today and in the nearer future, thus keeping in mind practical applicability.
- By validating the new solutions by discussing them with specialists in product development in all phases of the work and, as a result of considering the real needs in today's product development, by implementing software solutions following the ULEO approach and by applying them within several use cases (scenarios).

Consequently, this research's **major benefits for practical product engineering** could be characterized as follows:

- **Informational integration** within in a GIS enables process step applications to broaden their informational horizon, thus providing engineers with the information necessary for well-founded decision-making, as commonly known in the context of Design-for-X. This is applicable for OEMs and their suppliers. Additionally, change processes are put on a more solid foundation, as the links between the individual steps of product development are traced and managed by the system. Further, the existence of a single global information space avoids re-inventing solutions for solved problems, which is especially important for product development as it also reduces the variance in products and processes. Products can be manufactured more cost-saving, and processes and resulting products can easier be compared to each other.
- ULEO creates a framework for implementing the **automation** of routine tasks by offering a basis to represent the respective knowledge and its access by applications as well as a methodology to control inter- and intra-application automation. This increases the degree of standardization of methods and tools and improves the quality of products and processes by avoiding errors. Moreover, the GEOR method allows dynamic use of automation (as well as of other information), which allows for varying grades of details having to be handled by workflow management systems, making a redundant remodeling of product structures inside the latter superfluous.

Chapter 16 Final Discussion and Assessment of the Approach

In the following, ULEO's hypotheses shall be critically reviewed from what has been reached today, structured according to the requirements for IT solutions[*]*. Therefore, this section gives an overall validation of the ULEO approach, based on its above-described practical application: it sets out experiences and findings gathered during the prototypical implementation and application, also critically reflecting the outcomes of the solutions taken. For a better correlation to the research hypotheses stated in Part IV – 9.1, the concerned hypotheses are given in parentheses within the text. At the end of this chapter, the solutions found are compared to possible alternatives.*

16.1 Global Information Space

A method for the implementation of a **global information space** has been presented. It fulfills the above definition and information-technologically integrates both existing and new process step applications. The software that has been implemented following this approach proved that the ProSAps involved can share a **common information model** that holds the individual ProSAps' background information (semantic kernels – no unification of the terminology), as well as a company's know how (**hypothesis 2d**): all the informational entities can be correlated to each other by typed relationships, whose semantics is documented in the MTRT (hypothesis 3b); **semantic kernels** have been bridged where it was beneficial (hypothesis 2). Not implemented yet is the utilization of **OO interfaces** for the navigation through the GIS (hypothesis 3d), as this is one of ULEO's most advanced features, which will be applied after the more basic features will have been fully deployed. The consequence is the desired global information space that is brought to life by the representation format **ULEO XML**, a set of **GIS services** that are based on inter-process communication and that are able to transport all kinds of information the global information space can handle. Moreover, the implemented software showed that it can **react dynamically** to the contents of the global information space and change its behavior accordingly. For example, background knowledge modifies the set of available design features, and contents of proprietary product models serve as input for the automated creation of other vendors' downstream models (such as inspection programming) or their modification causes model-spanning synchronization.

Software add-ons to existing ProSAps for GIS integration. The **availability** of CAA API functions for accessing CATIA^{TM} files turned out to be sufficient for the current applications within quality assurance. As discussed in earlier chapters, this issue might become a practical **bottleneck** in other situations for opening up applications to the global information space. Yet, workarounds have been identified in this work. Within Dassault Systèmes' current software, the relations, in particular, are handled restrictively in terms of accessibility to external software. Also, they are technically reduced to certain pre-defined types. As the types of relations needed in the quality assurance domain, do not exist in Dassault software anyway, this proved not to be a problem: in the quality assurance domain, **EOR instances** facilitate the associativity between CATIA^{TM} EO instances stored proprietarily and neutrally-stored EO instances of the inspection planning application. As these relations are not managed by Dassault software[†], they are open to all interested applications and any desired kind of relationship could be introduced, managed, and used. The taken approach of storing EOR instances inside a **dedicated database** managed by the ULEO server proved to be suitable for practical application (**hypothesis 1b**). Any ProSAp could access any EOR instance needed. The representation of several kinds of strategies as **EORMs** inside UMEO yielded a similar effect towards a free global information space (**hypothesis 3g**). **Typification** of relations proved to be one of the

Footnotes

[*] As the latter are more fine-grained than the hypotheses.

[†] The necessary relation types do not exist in Dassault software.

basic prerequisites for such relation processing (hypotheses 3b+e). Inspection strategies could only be processed because they were correlated to other types within a taxonomy of relation types, **MTRT** (**hypothesis 3c**). The enforcement of a **strict encapsulation** when representing informational entities (**hypothesis 2a**) turned out to be the only realistic option when trying to convince vendors of existing software to plug into the GIS, as this harmonizes with their systems' object-oriented philosophy. Having to collect distributed IE descriptions or handling existential quantification would not be accepted. The philosophy of **relation-based navigation** (**hypothesis 3**), on the other hand, harmonizes well also with legacy software and with the nature of information in product development in general (high correlation, correlation with correlations). **Strict encapsulation** is a prerequisite of relation-based navigation.

It has also been pointed out that the method of managing EO instances within unchanged ProSAp databases depends on the existence of persistent but not necessarily unique **IDs** of EO **instances** within the ProSAps.

16.2 Information Structure

ULEO's GIS information structure (informational base types and models) proved to be sufficient for the scenarios investigated. Especially the advanced possibilities of modeling **relationships** (**hypothesis 3**) showed to be useful, as relations have been employed intensively within the domains addressed. They played the vital role of key informational elements within the quality assurance process (quality criteria, inspection strategies, etc.), not least because they constitute natural building blocks for many informational entities of the real world. The **typification of relationships** (**hypothesis 3b**) facilitated quick and reliable information retrieval. As currently no other application than the ULEO server[*] offers MEO services, the **external referencing** mechanisms could not be evaluated in depth – although they were implemented in a light version. This will be subject to future extensions of the system. The **explicit representation** and documentation of informational entities inside UMEO enables various kinds of **strategies** to be stored in a way that is focused and semantically clear. The description of the object- and relation classes used inside CATIA^{TM} V5 and relevant for being shared within the global information space has been entered into UMEO. Also in this case, the expressiveness was sufficient and intuitive modeling was possible.

ULEO's information structure provides low redundancy and high universality through separation of objects and relationships (**hypothesis 3a**) and through inheritance between classes of objects and relations. Adding new relationships allows utilization of existing IEs for new purposes and in new contexts. New types of objects and relationships can be introduced at any time and applications can use them either by reaching them via already known relation types or by interpreting the direct semantics expressed as meta-information. Attached **context** information (**hypothesis 2b**) allows any IE to be utilized in one or more contexts while preventing misconceptions occurring by the use of identical identifiers and avoiding the need to use a unique definition of a concept that occurs differently in other applications' views[†]. The GIS-wide **identification and addressing schema** turned out to be essential (**hypothesis 2c**). The flexibility in being able to handle any new type of concept or relationship is supported by the optional deployment of **standard atomic data types**[‡] used in attributes and methods to assure a basic understanding between GIS participants. Based on this, complex data types and concept- and relation types can be constructed according to the EOx structure defined for the ULEO GIS. Because of this possibility to understand the atoms of new EO classes and relation types, and because of the semantic descriptions, it is neither necessary nor useful to

Footnotes

[*] For accessing UMEO

[†] Same ID: two similar but different definitions; strictly viewed, there are two concepts.

[‡] Such as XML schema types

standardize complete types of concepts or relations. Consequently it is possible to **directly utilize proprietary IEs** and to integrate them within IIM/UMEO. The application to quality assurance proved this idea of integration by individualization instead of homogenization to be valid and highly beneficial. Hence, existing applications can cooperate using the original non-standard EOs, thus able to provide their full expressiveness and functionality to others. The **implemented ULEO software** demonstrated the usability in practice of new EO classes (extended help scenario, CATIA^{TM} UDFs, EXPRESS model import) and of EORMs (inspection strategies) derived from known types. The **context** concept is also the foundation for being able to store EO instances from several ProSAps in the central **EO instance database** (**hypothesis 2b**). The extended help scenario demonstrated the integration of complex **background knowledge** with specific information (**hypothesis 2d**). The light version of **external referencing** evidenced knowledge base-spanning integration of background knowledge. **Hybrid knowledge representation** within attributes and methods turned out to be of use for handling optimized (inspection strategies) or standardized (EXPRESS) representation formats within the ULEO framework, thus achieving greater expressiveness and usability (inspection strategies) or integrative potential (EXPRESS), respectively.

16.3 Inter-Process Communication Interface

The **ULEO inter-process communication interface** (GIS services) enables ProSAps to request and write information of the kind and amount needed at the time desired, thus avoiding redundant file-based information storage and its negative consequences. The **implemented** software proved the feasibility of this approach. In the scenarios, this turned out to be easily implementable, even within Visual Basic applications.

Inter-process communication interface: service concept. The decision to choose a service concept (**hypothesis 1a**) for the ULEO inter-process communication interface turned out to be powerful and flexible. The individual services had to be adapted in detail and were enhanced during the software implementation phase. This was primarily a result of granting ProSAps more comfortable access to the global information space.

ULEO XML. ULEO XML is optimized for the given targets and prerequisites such as balanced expressive power and runtime efficiency. The **expressiveness** of ULEO XML turned out to be sufficient. No cases where informational entities could not be represented straightforwardly showed up. The generation and especially the interpretation of XML code inside the ProSAps were made simple using public domain parsers and could be elegantly integrated into the individual object class methods' program codes. The usage of **XML** inside the ULEO **database**, which is hosting UMEO and MTRT contents, was the foundation for making changes of database formats unnecessary. During the software implementation phase, new EO- and EOR classes were introduced as new aspects that had not been considered in the specification phase arose from the application domain. As not only the UMEO- and MTRT tables within the ULEO database, but also the tables for EO instance and EOR instance information store their data using XML, and since they are designed for hosting any type of EOx, such changes succeeded very comfortably. Since XML is based on text format, it of course did not yield any problems concerning transferability between different software applications. XML as an add-on specification to plain text format is a powerful means of bringing structure into chunks of texts. XML schemas help to assure correct formats. Hence, the use of **XML** inside the **inter-process communication interface** turned out to be a good choice, especially as quite complex EOx information has to be transported. The efficiency aspects discussed in a previous section proved to be unproblematic in the scenarios surveyed.

16.4 Reuse and Automation

Reuse and automation have been **practically demonstrated** by inspection strategies (**hypothesis 3g**), feature constellations (**hypothesis 3i**), and part templates (**hypothesis 3h**) that can flexibly be created using any desired type of relationship and that can be – if desired – instantiated automatically by the use of GEORs (not implemented yet). Being a part of UMEO, such building blocks are part of the company's background knowledge and are therefore, in whole or in part, also usable by downstream ProSAps. Moreover, the suggested solution for reuse and automation is universally applicable to any domain and any kind of concept and relation. Application-spanning automation using **event handling** and **GEORs** (**hypothesis 3f**) could <u>not</u> be <u>practically validated</u> so far, as this is the second step after introducing the ULEO concepts.

16.5 Usability

The usability of ULEO-based software for the engineers is improved by three key aspects:

- Optimized EO classes within IIM/UMEO
- Provision of background information (see above)
- User views

Only the last issue has not been investigated yet in practice.

16.6 Flexibility, Scalability, and Other Requirements from Practice

Flexibility and scalability of the approach in the above-mentioned sense have been **practically demonstrated** by the variance of prototypical and productive implementations. They are further fortified by the variety of potential future application scenarios not implemented thus far.

The process of convincing OEM software departments to take over ULEO concepts proved **scalability** to be essential for practical applicability and acceptance. The consequences have been set out in detail; the most prominent ones shall be restated here: integrate existing commercial software as deeply as possible into new concepts (**hypothesis 1**) and start with less but continue with more functionality. From this, it is concluded that ULEO's scalability factors are very important.

Further pertinent comments on scalability are mentioned below:

- Achieve scalability by using as much functionality of existing software as possible and offering new functionality <u>additionally</u>.
- Avoid changing the data management of legacy software. As ULEO proposes to separate EOs from EORs (**hypothesis 3a**), this is easy to achieve as EOs do not know about their external relationships (although the applications are able to retrieve them, of course) and thus do not have to be modified if being used inside the GIS.
- As EORs may represent automation knowledge, the degree of automation is dynamically adjustable by adding such EORs (IEORs and optionally GEORs) into UMEO.
- As also other company background knowledge such as best practices is to be represented inside IIM/UMEO, any kind of such high-level information may be inserted into IIM/UMEO and deployed inside the applications according to the practical needs and strategies. Companies may start off with a very small IIM/UMEO and end up with ontology-like model contents. Thus, sophistication of knowledge representation inside IIM/UMEO is **scalable**, which permits both backward compatibility to older feature-based systems and also integration of and cooperation with company-wide information systems (**hypothesis 2d**). Hence, a varying complexity and a varying amount of IIM contents are supported as required.

✍ Right behind scalability and implementation costs in importance as a success factor is the **explainability** of a scientific concept.

Supplier integration has been shown to be achievable by means of integration into the GIS and file-based information exchange.

A basic methodology for **information acquisition** has been developed and practically investigated (see I++ and integration with other OEM-internal information models). A mixed top-down/bottom-up approach has been favored. The results proved its feasibility.

16.7 Comparative Study

This section contrasts the general hypothesis of this research work to alternative approaches and discusses their behavior if they would have been applied instead to the above-described scenarios.

Part (1) of the hypothesis. "Legacy applications keep their information processing and are opened up for online information sharing by code extension (e.g., via APIs) and their instance information is integrated by sophisticated relations".

Alternative approaches centralize the management of specific (instance) information. This must be considered not realistic in the scenarios considered, or at least much harder to realize, as it would cause deep changes of the applications involved potentially preventing them to remain usable.

Dismissing the integration of instance information through EOR instances is an option, but leads either to unconnected instance databases or would again require changing the applications deeply for managing such relations within their own data – however, this last solution would lead to problems concerning the access of relations and concerning the redundancy of information storage. Furthermore, relations could not be correlated by other relations.

Part (2) of the hypothesis. "Providing an online-accessible and -processable documentation of all kinds of information in the GIS by means of general information and meta-information. There is no unification of terminology, inconsistency is allowed and handled; semantic kernels arise".

Existing ontology-based approaches apply the same principle of providing a documentation of occurring types of instance information. However, there are crucial differences in how this is achieved and how it is applied. For example, ontology-based approaches support the principle of unifying terminology. In the scenarios reviewed, this would have required to synchronize many software projects in product development and to apply strong efforts in order to find a common terminology that is general enough to stay constant over several years. Both aspects must be judged unrealistic from a practical viewpoint, mainly due to the resulting costs and existing political fences. Such solution might have worked for integrating applications in the quality assurance domain, if the semantic kernel formed by the I++ information model would have been detailed to its maximum extent. This would have also meant to standardize all kinds of features for design and machining and all kinds of user-defined tolerances (the current I++ model allows adding and specializing entities in the model). This approach would (1) require a large amount of time, and (2) require to standardize processes amongst several automotive OEMs down to very small details, and (3) require the OEMs to publish important intellectual property. Discussions in the I++ working group judged such approach to be not desirable. As a result, ontologies using a unified terminology are actually intermediate models with a restrictive character (the terminology does not satisfy the needs of recent software developments and prevents direct relations). Such direct relations proved to be very useful in the quality assurance scenario, as they yielded an information model that was judged to be very natural by the I++ participants. The option to extend the semantic kernel formed by the I++ information model, was utilized several times by individual OEMs.

From what has been discussed so far, it can be concluded that the solution to use several domain ontologies and apply an ontology-mapping approach to integrate them is to be preferred. However, typical ontology mapping solutions neither provide sufficiently expressive relations, nor a sufficient context management for IEs, nor a GIS-wide addressing schema. Furthermore, mapping ontologies typically work on higher abstraction layers (general ontologies) and prevent direct relations between the domain ontologies from being created. Without context information, informational entities to be stored in a database have to obey restrictions concerning identification and consistency; such restriction proved to be undesirable in the practical scenarios (e.g., feature types of several domains share the same names, but are in fact different, i.e., inconsistent). Context information is also necessary for storing and retrieving instances of several applications in the same database. GIS-wide addressing proved to be crucial to access de-centrally stored instance information. Finally, as a mapping ontology basically uses relatively weak relations (such as "is_the_same_as" or "is_equivalent_to" or "is_similar_to") to map the concepts of the domain ontologies on each other, practically important relation types (e.g. strategy-holding ones) cannot be represented. If the mapping ontology works with a common, more abstract, terminology, it is another kind of intermediate model and yields the drawbacks, described above for unified terminologies, and prevents direct and powerful relations to be used between the domain ontologies – this is not surprising, as a mapping ontology is to map ontologies and not to represent the knowledge[*] lying in-between the ontologies.

Other alternatives to document the information types processed by the applications are to utilize a documentation that is not online-accessible, and to employ exchange formats such as STEP, that have to be hard-coded into the applications. Also the typical Corporate Information Systems don't use online documentation of concept types. For both exchange formats and CIS applies again what has been stated about intermediate models. For the quality assurance scenario, the expressivity of the existing STEP APs was not sufficient. This fact has of course already been recognized by researchers who strive for implementing a solution for quality assurance.

The alternative, not at all to document the information types is not suited for an information exchange solution that is to be able to universally serve any kind of application.

Part (3) of the hypothesis. "Introducing sophisticated relations in the GIS and applying relation-based navigation for integrating all kinds of informational isles. Furthermore, these relations support automation in terms of storing strategy information and constellations. They represent building blocks as constellations. Relations of specific types control the flow of automation."

The relevance of sophisticated relations in the described sense has been proven in the practical information modeling work, for instance, in the I++ working group. As has been motivated, there seems to be no comparable alternative regarding the naturalness of the resulting models. Nevertheless, less powerful relations can be applied in spite of this fact, which leads to less natural information models and the lack of options for relation-based navigation. Although in principle also possible for simple relations, the navigation would obviously lose much of its universality. This fact would, in turn, lead to less flexible and less "future-save" applications. As already stated, this has not yet been verified in practice. A set-based information access has already been identified to be of relevance in certain domains and/or situations (information needs), but this did not occur in the analyzed scenarios. Typically, only one set-based retrieval operation was performed right at the beginning of a retrieval sequence. This set-based retrieval could be formulated as "give me all instances of EO type X" and is supported by the current implementation of GIS services, in that a wildcard may be used for specifying the EO identity.

Alternatives for storing strategies are, for instance, knowledge bases and individual script files. Although the knowledge maintenance aspect has not yet been evaluated in practice, the access to the EOR-based strategies within UMEO proved to be natural and robust, and required no further overhead for managing of this knowledge.

Constellations, building blocks, and GEORs can in principle also be realized with simple relations. However – depending on their exact expressivity – the option to nest them might not exist and the knowledge stored within the sophisticated relations might have to be implemented inside the applications using them. Thus, again the flexibility and universality of the using software decreases. The practical need for offering constellations and building blocks to the engineers has been shown in the practical scenarios. GEOR support has not yet been implemented (see also the end of Chapter 15 for details).

Footnotes

[*] Such as strategies

Chapter 17 Final Overall Assessment of What Has Been Achieved and Not Achieved

Following the detailed evaluation, this section condenses the insights and lessons learned.

The major concepts of ULEO have so been tested and validated – please refer to the end of Chapter 15 for the exceptions. In respect to the **costs** for implementing ULEO, the experiences from the practical scenarios showed that the **benefits** offset the implementation costs in all cases. Scientific discussions and practical applications seem to confirm the validity of the proposed solutions in principle. Nevertheless, as many decisions have been made considering detailed interpretations of the major concepts, individual ones will certainly be subject to future improvements.

ULEO is **implementable** and **applicable**. It has been pointed out that most, but not all of ULEO's features, could be **validated** in practice. Those that were judged sound proved to be widely applicable without major problems (the aspects to keep in mind have been mentioned). From the current state of experience, ULEO could improve product development. OEM officers expect major benefits and savings from the ULEO-based software system and intend to extend its deployment to at least several automotive plants within Germany. Engineers who know about the new possibilities from former interviews have been asking for its early introduction.

Nevertheless, it is important to note that this practical applicability is based on intensive **research work**. Problems have been identified, the state of the art has been investigated, own solutions have been developed and formulated, and the scientific communities have been involved in discussing them by attending conferences and numerous personal discussions. The already tackled applications of the ULEO approach indicate that its solutions are as of interest in scientific projects as they are in OEM-internal projects that are closer to practical application.

Due to ULEO's holistic nature resulting from the combination of scientific base work and the consideration of practical constraints, the ULEO approach is assumed to be not restricted to product development applications as will be discussed in the recommendations section.

Chapter 18 Update on the State of the Art

After the initial literature studies performed in this work, other attempts have been started to achieve related targets. In the meantime corresponding results are available. This chapter highlights the most relevant of these endeavors and identifies relations to and consequences for this work.

18.1 New Developments

This section briefly surveys the most relevant efforts outside this work.

Since the beginning of this research, several doctoral theses have been finished within the same scientific environment. Two of them are based on the above-discussed work of Eric Lutters (see *[Lutters, 2001]*, section Part III – 7.1.2): Rob Mentink suggests a solution for workflow management, and Dirk Wijnker develops the concept of interface ontologies (see *[Mentink, 2004]*, *[Mentink et al., 2002]*, *[Wijnker, 2003]*, and *[Wijnker et al., 2000]*). Alexander Layer has been concerned with the design-concurrent estimation of manufacturing costs utilizing case-based reasoning techniques; his work resulted in the CABACO approach (case-based cost estimation, see *[Layer, 2003]*, and *[Layer et al., 2001]*).

To reach his goals, Rob **Mentink** suggests further means for the integration of multiple information models and the respective pools of specific information* in that he introduces the concept of a top-level *environment ontology* and built-in inheritance relations. Thus, he achieves a wider information space as a basis for workflow management. Basically, also Dirk **Wijnker** performs a mapping of information sets through another ontology that he calls *interface ontology*. An interface ontology, however, serves the purpose of plugging an existing application into the information management system without having to hard-code the knowledge relating to the IEs to be mapped onto each other. **Discussion**. In the existing ULEO implementations, interface ontologies do not exist; yet, they do have a place in the suggested ULEO architecture as they would be applicable in such cases where semantic kernels exist in IIM and many ProSAps are willing to relate to it. This could pay off in the discussed field of quality assurance, for example. Thus, the implementation of a functionality as that suggested by Dirk Wijnker will be considered as future work in the context of standard ULEO GIS API production. The principle methodology of Rob Mentink could be of interest in combination with the ULEO approach. However, the information structures it is based on are different, as ULEO's GIS has been developed according to the very combination of targets and prerequisites.

Alexander Layer tackles the field of Design-for-X, i.e., widening engineers' informational scope. Design-for-X strives to support product designers with such information, allowing them to consider aspects of downstream processes during their design, thus achieving a satisfactory design earlier with fewer feedback loops. **Discussion.** Alexander Layer's CABACO approach is fully compatible to ULEO, not least due to an intensive exchange of views between the research fellows. Figure 47 illustrates the similarity. Amongst other findings, the thesis *[Layer, 2003]* identifies a series of EOR types of practical relevance for cost estimation. Hence, implementation of a CABACO solution for the GIS is on the list of future developments.

Footnotes

* Called *information structures* in Eric Lutters's terminology and affiliated to orders, products, and resources

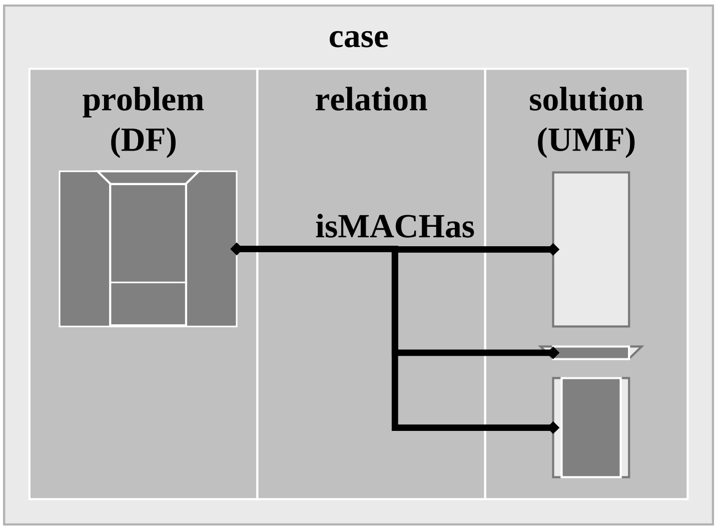


Figure 47: Case in CABACO [A. Layer]

✍ As mentioned during the discussion of *[Lutters, 2001]*, the research of Eric Lutters and the author’s research are founded on deviating sets of targets and on deviating basic prerequisites and assumptions about information processing around and about engineering. As was to be expected, the developed solutions deviate accordingly, which again confirms the insight that the quality of any solution depends on its destiny and there are no truly generally applicable solutions. Nevertheless, solutions based on and built on top of rather fundamental approaches such as Information Management and ULEO can, in principle, make use of any of them, which facilitates a clustering of the research work.

Another research, which is still ongoing, is concerned with a so far unpublished scientific approach called *interconnection management*[*], or in short ***Interconnections***. Marco Groll has the same affiliation as the author and is investigating a new methodology for product documentation that puts relationships between objects in the center of its interest. This methodology replaces the traditional representation of product structures by a new and less hierarchical one, promising to enable significantly more powerful deployment of this information. **Discussion**. The Interconnections approach does not directly deal with the integration of applications or with representation formalisms. In fact, it matches well with the ULEO approach, as ULEO can provide an integrative foundation on which the Interconnections ideas can build.

The **Product Engineering** program (**PE**) of the *Manufacturing Engineering Laboratory (MEL)* of the *National Institute of Standards and Technology (**NIST**)*, USA, tries to “*establish a semantically-based, validated, **product representation scheme** as a standard that supports the seamless interoperability among current and next generation computer-aided design systems (CAD) and between CAD and other systems that generate and use product data, to help the manufacturing industry achieve a 10 percent reduction in interoperability costs and a tenfold improvement in design exploration capability.*” (NIST program *Product Engineering*[†]). In addition to interoperability of systems, the program targets collaborative working among distributed designers and design teams, integration of data and knowledge across the product development cycle (from design to analysis to manufacturing and beyond), as well as knowledge capture, exchange, and reuse (see *[Fenves et al.,*

Footnotes

[*] Originating in the German term *Verbindungsdokumentation*
[†] See the URL http://www.mel.nist.gov/proj/pe.htm.

2003]). A future PLM[*] system shall be able to directly access relevant information without having to rely on a segmented architecture of PDM and CAx systems. Thus, according to the program members, design rationales and reasons for changes will be accessible more easily. Also, PLM systems will be able to cover very early stages of product development where no detailed geometry is as yet available. **Discussion**. PE's program goals widely overlap with the goals of this work, thus confirming their relevance. However, from what has been achieved thus far by the program members, it can be concluded that the solutions are or will be different in respect to several significant issues – primarily since they rely on STEP's principal philosophy. To illustrate this, some issues are sketched:

~ ULEO's Integrated Information Model IIM+MTRT is dynamic and accessible online instead of fixed and pre-defined before system startup, as is the case with PE and STEP.

~ PE defines, as ULEO does, a taxonomy of relationships used inside the core- and the derived models and represents relations as classes, emphasizing the importance of relationships for representing engineering information. However, since the set of PE's relations are pre-defined, no background knowledge, which is relation-intensive and dynamic by nature, can be integrated into the PE ontology. Furthermore, relations between relations do not seem to be allowed in the PE approach, while the author argues that they are crucial for natural and compact modeling.

~ PE does not make use of the concept of contexts in the ULEO sense, which enable an integrated handling of knowledge from the various domains and a multi-facetted and non-unified use of concepts. Furthermore, context handling is a prerequisite for maintaining local consistency.

~ PE has not yet developed an interface for implementing the cooperation framework. A vertical API is planned to permit PLM systems to access ProSAp information; the horizontal information exchange between (potentially heterogeneous) ProSAps still remains to be investigated and developed. These goals have been identified within the PE program. In this respect, the author advocates the ULEO inter-process communication interface as described above.

To **sum up** this discussion, PE's and ULEO's goals are partially similar, and some interesting perceptions have been made in both PE and ULEO. This may indicate a new trend for future IT solutions for product development: relations are crucial, semantics have to be clear for a sophisticated information flow, and a broad exchange of information on a standardized basis is desirable. However, the solutions differ significantly, mainly because PE relies on STEP's underlying philosophy, which has been argued to be inadequate for reaching the stated goals. It is therefore concluded that PE does not add key insights to this research but rather confirms the relevance of the targets.

The OMG's **Model Driven Architecture** (**MDA**)[†] (see also *[OMG MDAex*, *2005]* for an executive overview) is currently being developed and promises a new, flexible approach to software engineering. Being a future standard, this approach promises to be well tool supported. The three primary goals of MDA are portability, interoperability, and reusability of software through architectural separation of concerns. It is therefore suggested that the specification of the operation of a system be separated from the details of the way the system utilizes the capabilities of its platform. In a sense, MDA is comparable to UML for information modeling but targets system specification, instead. Software systems are specified by models in specialized languages. **Conclusions in short**. OMG is potentially beneficial for implementing commercial software. Its practical applicability has to be evaluated; this is, however, not the focus of this work. The application of MDA for designing and implementing a ULEO server, for instance, would combine too many new approaches to be able to sufficiently judge the impacts of individual design solutions.

Of course, there is also new functionality available in **commercial software** systems. However, these approaches, as far as they could be surveyed within this research, do not provide significant new

Footnotes

[*] Product lifecycle management

[†] See the URL *http://www.omg.org/mda/*.

concepts. Instead, they utilize existing concepts and technology to provide new functionality. This is presumably the case, as there are still very many practical problems to be solved by these systems such as runtime efficiency, stability, and usability. It is therefore **concluded** that these systems' limitations, which have been stated above, still exist.

18.2 Consequences for This Work

What are the consequences of the preceding section's discussions on ULEO?

To sum up the insights gained within the preceding section, new developments tend to confirm the targets of this research and suggest some future work concerning the support of practical implementations and their application to new fields in product development and on company-wide workflow management. Although there are alternative approaches, it is assumed that they do not question this research's hypotheses.

Chapter 19 Recommendations and Future Work

What has been contributed by this research will flow into further research tasks and development projects in the future. This chapter outlines the most relevant issues.

19.1 Recommendations

There certainly is a large collection of issues concerning details of the ULEO approach and its environment to be further explored and specified. Some of them are identified in the subsequent section on future work. The current section focuses on more fundamental issues of IT for engineering.

Costs and benefits. It has been stated above, that for the ULEO implementations realized until now, the benefits exceeded the costs in all cases; nevertheless, there is one aspect that is to be considered closely for future implementations: the degree of using **company know-how** (background knowledge). Strategies reflecting automatable routine* tasks in engineering promise to pay off if added to the machine-processed set of knowledge. However, complex and/or infrequently applied knowledge is to be judged as critical by default, as it tends to be hard to formalize and maintain, and as it often ages faster than simpler knowledge. This includes knowledge that can be used to assess engineers' work but also "intelligent" help information, as expert systems used to offer. The recommendation that can be given here is, to estimate the costs and the savings as carefully as possible.

Having made positive experiences with the acceptance of the ULEO approach from IT responsibles and implementers at a major automotive OEM and within the OEM-spanning I++ workgroup (see above), the author would like to recommend the following to other researches: it pays off not to forget the practical restrictions when looking for new concepts that are supposed to be applied in a few-years range of time.

The human factor and the consequences of a technically ideal world. Science is still far away from having developed what are generally called intelligent machines. Man is still the one that adds intelligence to processes and thus really does the work. Current approaches help to make these processes more efficient and effective, i.e., to raise the **productivity** of the people and systems involved. But what can be the **side effects**? Fewer engineers will be needed to fulfill the tasks in product development. On the other hand, the remaining experts will have to be highly skilled, as the typical informational small-zones are opened up, and engineers will have to consider aspects from several steps in product development. In part, they will have to lay the informational foundations for subsequent engineers. In all, their work will become more interesting and multi-faceted. It is not within the scope of this work to suggest how to employ those people who are not needed anymore – as this is a very common problem, society as a whole is called on to solve it.

Possible fences – the psychological factor. Assume that all these goals have been reached. **Automation** can cause situations where users cannot follow and understand a system's behavior. This might lead to increased bias and keep system errors from being discovered. Consequently, **explanatory components** are an increasingly important aspect of automation and should be considered from the very beginning. ULEO's information structure supports this method. Furthermore, automation has to take proper prerequisites into account. This means that automation may not be used under all circumstances. Hence, complete trigger conditions have to be part of the automation knowledge or users might have to be able to decide this. Generally, **acceptance** of new technologies by users can be critical, if the way of work changes rapidly and dramatically. This point again emphasizes the need for scalable systems. As with automation, all kinds of information offered

Footnotes

* Limited complexity, frequent usage

by a system that are bare of **meta-information** on its sources, age, reliability, and so on, nurture doubts. This aspect suggests a need to manage and display the respective meta-information.

A global information space accompanied by an intensive flow of information could **overwhelm** users. This suggests that the information packages presented to the user should be tied in carefully. User interfaces are a means of offering information in a multi-layered fashion, where details are given only upon being explicitly requested.

A brief note on practical information modeling. Experiences from practical information acquisition and modeling (see the section *Process of Filling UMEO – Information Modeling*) support Socrates' claim: "*I know, that I don't know anything – and even that I hardly know*". This implies that people should be critical of their own skills and knowledge and should question well-known things. But the statement can also be extended to mean use of a clear and generally understandable language, thus providing a wide and solid base for discussions. This includes clarifying notions before using them.

19.2 Outlook on Future Work

This outlook examines several aspects: the current implementations are complemented, new application fields are explored and worked out, and further research to be done to generate and utilize further fundamental findings is sketched.

Prototype Functionality to be Implemented

The existing prototypical and productive implementations based on the ULEO approach are intended to be completed by additional functionality. This includes thus far unimplemented ULEO concepts but also practically useful technical details.

Not yet implemented and thus subject to further completion projects are the features listed below:

~ Table of information representation formalisms (TOIRF)
~ Table of identifiers
~ Table of semantic description languages (TOSDL)
~ UML conversion
~ Support of OO interfaces in relation types
~ Support of GEORs
~ ULEO admin: consistency and redundancy checks are currently being implemented with means for handling archiving and versioning in the pipeline.
~ Improvement of user management and access rights and SSL (secure socket layer) encryption of data transfers over the network
~ Freeware library for GIS integration of existing ProSAps that provides code for using and offering ULEO IPCI services under consideration of Dirk Wijnker's insights (see *[Wijnker, 2003])*
~ Extension of GIS services by set-oriented information retrieval services
~ Integration of constraint solvers, e.g., for interpreting EORM contents, in line with possible solutions developed in *[Salomons, 1995]*[*] and shown in *[Leemhuis et al., 2002]*

Footnotes

[*] The concurrent engineering tool FROOM (Features and Relations in Object Oriented Modeling) supports assembly modeling, tolerancing, and design history (reuse). See also *[Salomons et al., 1993]*.

Future Application Fields

Apart from the application fields and scenarios already elaborated, a variety of further fields will be more closely considered in the future. Some of them are sketched here.

Design-for-X will certainly be a major concern when applying ULEO in the future. This includes application of the CABACO approach (see *[Layer, 2003])*. Process planning will continue to be of relevance as well: contributions to the European MODALE research project have recently been made and will be sustained. Finite Element Meshing strategies are of special interest as there is no appropriate management for them at the moment.

Further, the entire range of **upstream steps** of product development is of major relevance as these steps shape the product and its predecessors such as requirements and functionality. In this context, computer-aided conceptual design as introduced by Krause et al. (see *[Krause et al., 2003]*) contributed major findings that might be of value if integrated with the ULEO approach – the current **iViP** integration infrastructure (see *[Krause et al., 2001 and 2002]*) is based on PDM enabler and CAD services[*], both of which have been discussed in this thesis. They could be replaced by ULEO GIS services. Computer-aided conceptual design tackles inter-dependencies between requirements, functions, and components.

✍ Due to the topic’s relevance, the following quotation shall briefly provide the background information on iViP: “*As part of the German research project for integrated Virtual Product creation – iViP, funded by the German Federal Ministry of Education and Research [Krause et al., 2002], a software solution for supporting the design process in the early phase has been developed. As product functionality was the centre of modelling support, the system was named Function Oriented Design – FOD. FOD provides support in the major conceptual phases. The assistance system consists of a set of separate editors for (a) Setting up requirements; (b) Specifying functions and function structures; (c) Defining the part structure and its components and (d) Managing nets of parameters and constraints;*”

There is a good chance that, in principle, not only is ULEO applicable within the field of product engineering, but that it can also be considered a core methodology for **integrating applications throughout a company**. This originates primarily in the consideration that the assumptions made about existing software applications and their co-operation are valid for the huge majority of IT systems deployed within companies. Also, the targets that influenced ULEO’s development can be assumed to be generally valid in this area of IT systems. It is hence considered promising to integrate software of the logistics process chain with IT for engineering. In this context, consideration of Rob Mentink’s findings is also conceivable (see *[Mentink, 2004]*).

Further Research

This section, which does not claim to be complete, spotlights some research tasks to be tackled in the nearer future.

The exploration of various forms of **information acquisition** (knowledge acquisition, e.g., for filling UMEO) is currently being tackled in cooperation with the Ruhr-University in Bochum, Germany. Of special interest is the application of ULEO for the representation of highly complex **engineering**

Footnotes

[*] “*The underlying interface is based on the OMG CAD services specification. At present, interfaces to CADdy++ Maschinenbau®, Mechanical Desktop®, SolidWorks®, Inventor®, Pro/ENGINEER®, are available.*” (see *[Krause et al., 2003]*).

templates and management of the interrelations between them. The same holds true for the principle positioning of current **PDM systems** and the GIS. In this context, the *Interconnections* approach (see above) is also of relevance. *Axiomatic design*, especially in combination with Christine Ping's *Target Cascading* approach, is a candidate to support designers by guiding the **design methodology** and is well suited to be implemented using ULEO (see *[Suh, 1990, 1995, 2001] and [Ping, 2002]*). Last but not least, *design by least commitment*[*] and concurrent engineering are targeted research fields in the future.

Concurrent engineering (CE), including the term's interpretation in the sense of simultaneous working, can be facilitated within the ULEO approach by letting engineers work in parallel and having the ProSAps log the individual EO instantiations. One of several feasible scenarios is set out briefly in the following. It is the subject of the author's further research. If an EO instantiation has an impact on other product models for which the initiating user is not responsible, the central automation management will send the system's automatic changes as proposals that have to be accepted or rejected with comments by the recipient. Thus, users are not able to directly manipulate other users' models, and changes will not be lost. Further, all users can do their work as usual, benefiting from automated product generation. This method of realizing concurrent engineering seems promising, since it offers engineers more freedom in terms of deciding when to do what, while insuring correct processing of product model changes. Also, the work of Rob Mentink will be of relevance here (see *[Mentink, 2004]*).

As has been pointed out, the engineer as a human being has to be moved to the center of any IT system's interests when trying to boost productivity. Thus, approaches from the scientific community of ***user modeling*** will be considered more closely. Elaine Rich's stereotypes (see *[Rich, 1979 and 1979b]*) are classical representatives of this research field, which searches for solutions to create and maintain models of humans inside computers. These models are supposed to cover skills, preferences, and other properties of the system users. User modeling systems include the dialog system PRACMA (see *[Jameson, 1994]*) and the user-modeling shell system BGP-MS[†] (see *[Kobsa & Pohl, 1995]*. In order to support sophisticated user modeling, information representation also has to adapt. This suggests that ULEO's solution of **hybrid representation** be exploited: issues become relevant, e.g., modality, reliability, and certainty and uncertainty of information. This kind of information processing is also called ***soft computing***. Uncertainty, which is frequently inherent to user models, can be handled by using methods of uncertain reasoning as examined by Kruse et al. (see *[Kruse et al., 1991]*). Anthony Jameson gives an overview of uncertainty management in *[Jameson, 1996]*. Herzog (see *[Herzog, 1994]*) has applied Fuzzy Set Theory (c.f. *[Zadeh, 1979]*) to user modeling issues. Jennings and Higuchi worked with neural networks (see *[Jennings & Higuchi, 1993]*). Thus, it might be promising to additionally consider soft computing for future product engineering solutions – not only for user modeling, but generally, since any piece of information inherently brings with it some uncertainty, vagueness, and modality.

As already mentioned during the discussion of the state of the art, the applicability of **deductive databases** for performing certain reasoning tasks, e.g. provided via service applications, is another point of interest in the future.

Finally, the integration of **case-based reasoning** techniques into the existing ULEO approach could represent an interesting complement to the explicit representation of information within IIM/UMEO (see, for example, *[Aamodt, 1995]* for an investigation on the role of case-specific knowledge and *[Layer, 2004]* for an application targeting cost estimation).

Footnotes

[*] For instance, ULEO may support re-instantiation.

[†] The Beliefs and Goals and Plan Modeling System provides user modeling capabilities to productive applications.

☞ The above explanations on the author’s further research topics concluded the moderated part of this thesis. They may have illustrated the large variety of challenges still left to be tackled on the way to sophisticated IT support for engineers in product development.

Part VII – APPENDICES

Chapter 20 References

All references cited from the Internet were revisited in January 2005 to insure the continued availability of contents.

Aamodt, 1995 Aamodt, A. (1995): "Knowledge acquisition and learning by experience – The role of case-specific knowledge"; In: *Machine Learning and Knowledge Acquisition: Integrated Approaches*; Tecuci, G.; Kodratoff, Y. (Eds.); Academic Press, New York; pp. 197-245.

Ananthanarayanan & Addala, 2002 Ananthanarayanan, Anand; Addala, Bharat (2002): "Definition and Usage of Combined and Assembly Features to Support The Concurrent Engineering Process Based on the Scenarios 'Cylinderhead Bolting' and 'Hole with Boss'"; master thesis; FHTE Esslingen (Germany).

ANSI, 1998 American National Standards Institute (ANSI) (1998): "Knowledge Interchange Format"; Draft Proposed American National Standard (dpANS) No. NCITS.T2/98-004; Document available online at URL http://logic.stanford.edu/kif/dpans.html.

Böhle & Rose, 1992 Böhle, F.; Rose, H. (1992): "Technik und Erfahrung – Arbeit in hochautomatisierten Systemen"; Campus Verlag, Frankfurt/New York.

Booch, 1994 Booch, Grady (1994) : "Object-Oriented Analysis and Design with Applications"; Benjamin/Cummings Publishing Company Inc., Redwood City, CA (U.S.A.).

Bozsak et al, 2002 Bozsak, Erol; Ehrig, Marc; Handschuh, Siegfried; Hotho, Andreas; Maedche, Alexander; Motik, Boris; Oberle, Daniel; Schmitz, Christoph; Staab, Steffen; Stojanovic, Ljiljana; Stojanovic, Nenad; Studer, Rudi; Stumme, Gerd; Sure, York; Tane, Julien; Volz, Raphael; Zacharias, Valentin; (2002): "KAON – Towards a Large Scale SemanticWeb"; In: *EC-Web 2002, LNCS 2455*; Bauknecht, K.; Tjoa, A M.; Quirchmayr, G. (Eds.); Springer-Verlag Berlin Heidelberg; pp. 304–313.

Bronsvoort et al, 2001 Bronsvoort, W. F.; Noort, A.; van den Berg, J.; Hoek, G. F. M. (2001): "Product development with multiple-view feature modelling", Proceedings FEATS 2001.

Chaudri et al., 1998 Chaudhri, V.K.,; Farquhar, A.; Fikes, R.; Karp, P.D.; Rice, J.P (1998): "OKBC: A programmatic foundation for knowledge base interoperability". In: *Fifteenth National Conference on Artificial Intelligence (AAAI-98), Madison, Wisconsin*; AAAI Press / The MIT Press.

Clément et al., 1998 Clément, A.; Rivière, A.; Serré, P.; Valade, C. (1998): "The TTRS : 13 Constraints for Dimensioning and Tolerancing"; In:, *Geometric Design Tolerancing : Theories, Standards and Applications*; Chapman & Hall, ISBN 0-412-83000-0; pp. 122-131

Dankwort et al, 1997 Dankwort, W. (1997): "Innovative Produktentwicklung – mit oder trotz Features"; In: *VDI-Berichte 1322 – Features verbessern die*

Produktentwicklung / Integration von Prozeßketten; pp. 331ff.

De Kraker, 1997 De Kraker, K. J. (1997): “Feature Mapping for Concurrent Engineering”; Doctoral Thesis; Delft University of Technology, Delft (NL);.

De Vin, 1994 De Vin, L.J. (1994): “Computer Aided Planning of Bending Operations for Sheet Metal Components”; doctoral thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twenty, Enschede (The Netherlands).

De Vries, 1995 De Vries, J. (1995): “A Process Planning System for Sheet Metal Part Processing – an integrated approach”; doctoral thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twenty, Enschede (The Netherlands).

DIN 32869-3-4 Deutsches Institut für Normung: “DIN 32869, Technical product documentation – Three-dimensional CAD-models”; 3 parts.

Doan et al., 2002 A. Doan, J. Madhavan, P. Domingos, and A. Halevy (2002): “Learning to map between ontologies on the Semantic Web”; In: *Proceedings of the World-Wide Web Conference (WWW-2002)*

Duineveld et al., 2004 Duineveld, A. J.; Stoter, R.; Weiden, M. R.; Kenepa, B.; Benjamins, V. R. (2004): “WonderTools? A comparative study of ontological engineering tools” (DRAFT-version), Dept. of Social Science Informatics, University of Amsterdam, The Netherlands; Document available online at URL http://hcs.science.uva.nl/wondertools/html/paper.htm.

Dürr et al., 1997 Dürr (1997): „Methoden der künstlichen Intelligenz in der featurebasierten Konstruktion und Arbeitsplanung“; In: *VDI-Berichte 1322 – Features verbessern die Produktentwicklung / Integration von Prozeßketten*; pp. 83ff.

Emmerich et al, 1999 Emmerich, W.; Finkelstein, A.; Schwarz, W. (1999). “Markup Meets Middleware”; In: *7th Int. Workshop on Future Trends in Distributed Systems (FTDCS99), Capetown, South Africa*; IEEE Computer Society Press; pp. 261-266.

Emmerich, 2000 Emmerich, W. (2000): “Software Engineering and Middleware: A Roadmap”; Invited Talk; In: *The Future of Software Engineering*; Finkelstein, A. (Ed.); ACM Press; pp. 117-129.

Farquhar et. al., 1997 Farquhar, A.; Fikes, R.; Rice, J. (1997): “The Ontolingua server: a tool for collaborative ontology construction”; In: *International Journal of Human-Computer Studies, 46;* pp. 707-727.

Fenves et al., 2003 Fenves, S. J.; Sriram, R. D.; Sudarsan, R.; Wang, F. (2003), “A Product Information Modeling Framework For Product Lifecycle Management”; International Symposium on Product Lifecycle Management, July 16-18, Bangalore, India.

Frank, 2002 Frank, Ulrich (2002): “A Multi-Layer Architecture for Knowledge Management Systems”; In: *Knowledge Management Systems: Theory and*

Practice; Barnes, S. (Ed.); Thomson Learning; pp. 97-111.

Fridman Noy et al., 2000 Fridman Noy, Natalya; Fergerson, Ray W.; Musen, Mark A. (2000): “The knowledge model of Protégé-2000: combining interoperability and flexibility”; Stanford Medical Informatics, Stanford University, Stanford, CA.

Fulton, 1992 Fulton, J. A. (1992): “Enterprise integration using the semantic unification meta-model”; In: *Proceedings of the 1st International Conference on Enterprise Integration Modeling*; Petrie, C. J. Jr. (Ed.); pp. 278-289.

Geelink, 1996 Geelink, R. (1996): “Flexible definition of form features”; Doctoral Thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twente, Enschede (NL)

Gorlen et al, 1990 Gorlen, Keith; Orlow, Sanford M.; Plexico, Perry S. (1990): “Data Abstraction and Object-Oriented Programming in C++”; Wiley & Sons, West Sussex, England; ISBN 0-471-92346-X

Grabowski et al., 1994 Grabowski, Hans; Anderl, Reiner; Erb, Jens; Polly, Adam (1994): “STEP – Grundlage der Produktdatentechnologie – Teil 1: Aufbau und Entwicklungsmethodik“; In: *CIM Management Vol. 10 Nr. 4*; R. Oldenbourg Verlag GmbH, München; pp. 45-51.

Haasis, 1995 Haasis, Siegmar (1995) “Wissens- und feature-basierte Unterstützung der Konstruktion von Stirnradgetrieben unter besonderer Berücksichtigung des Gußgehäuses”; doctoral Thesis; Fakultät für Maschinenbau und Verfahrenstechnik, Technische Universität Chemnitz-Zwickau (Germany); In: *VDI-Fortschrittsberichte Reihe 20*, Düsseldorf; VDI-Verlag, Düsseldorf (Germany)

Haasis, 1997 Haasis, S. (1997): “Nutzenpotentiale der durchgängigen Feature-Verarbeitung“; In: *VDI-Berichte Nr. 1322*; VDI-Verlag: Düsseldorf 1997; pp. 63-82.

Haasis et al., 2003 Haasis, S.; Frank, D.; Rommel, B.; and Weyrich, M. (2003): “Feature-based integration of product, process and resource”; In: *Feature based product life-cycle modeling*; Olling, G.J.; Soenen, R. (Eds.), Kluwer, Boston; pp. 93-108.

Han & Requicha, 1997 Han, J. H.; Requicha, A. A. G. (1997): “Modeler-independent feature recognition in a distributed environment”; In: *Computer Aided Design report, 30/6*; pp. 453-463.

Haugeneder & Trost, 1993 Haugeneder, Hans; Trost, Harald (1993): “Beschreibungsformalismen für sprachliches Wissen”; In: “Einführung in die künstliche Intelligenz“; Goerz, G. (Ed.); Addison-Wesley, ISBN 3-89319-507-6; pp 372 – 424.

Herzog, 1994 Herzog, Christian (1994): “Fuzzy-Techniken für das Verstehen von Studentenlösungen in intelligenten Lehrsystemen”; In: *Beiträge zum 7. Arbeitstreffen der GI-Fachgruppe 1.1.5/7.0.1*, “Intelligente Lehr-/Lernsysteme“ am Forschungsinstitut für anwendungsorientierte Wissensverarbeitung (FAW) Ulm; Gunzenhäuser, R.; Möbus, C.; Rösner, D. (Eds.).

Hunting & Park, Hunting, S.; Park, J. (2002): “Topic Maps and Global Knowledge Interchange

2002 – A Brief History of the Topic Maps Paradigm”; Document available online at URL *http://www.informit.com/articles/article.asp?p=29014&seqNum=5*.

Ishii & Miller, 1992 Ishii, K.; Miller, R. A. (1992): “Design representation for manufacturability evaluation in CAD. Beyond feature-based design”; ASME 1992; pp. 37-44.

ISO SUMM, 1991 ISO/IEC JTC1 SC2 WG3 (Ed.): “SUMM, Semantic Unification Meta Model”; N1360; 1991-Oct-15

ISO, 1994 ISO – International Organization of Standardisation (Ed.): “ISO 10303 Teil 1: Überblick und grundlegende Prinzipien”; In: *Reihe: Industrielle Automatisierungssysteme und Integration – Produktdatendarstellung und -austausch (Industrial Automation Systems and Integration – Product Data Representation and Exchange; vormals STEP – STandard for the Exchange of Product model data)*; Beuth Verlag GmbH, Berlin; 12/1994

Jaluria & Lombardi, 1991 Jaluria, Y.; Lombardi, D. (1991): “Use of expert systems in the design of thermal equipment and processes”; In: *Research in Engineering Design*, 2.; pp. 239-253

Jameson et al., 1994 Jameson, Anthony; Kipper, Bernhard; Ndiaye, Alassane; Schäfer, Ralph; Simons, Joep; Weis, Thomas; Zimmermann, Detlev (1994): “Cooperating to Be Noncooperative: The Dialog System PRACMA”; In: *KI-94: Advances in Artificial Intelligence*, Nebel, Bernhard; Dreschler-Fischer, Leonie (Eds.); Springer Verlag; pp. 106-117

Jameson, 1996 Jameson, Anthony (1996): “Numerical uncertainty management in user and student modeling: An overview of systems and issues”; In: *User Modeling and User-Adapted Interaction, 5*; pp. 193-251.

Jeckle, 2004 Jeckle, Mario (2004): “OMG’s XML Metadata Exchange Format XMI“; In: *Proceedings of XML4BPM 2004 – XML Interchange Formats for Business Process Management – 1st Workshop of German Informatics Society e.V. (GI) in conjunction with the 7th GI Conference “Modellierung 2004“*, Marburg; Germany; Nüttgens, Markus; Mendling, Jan (Eds.).

Jennings & Higuchi, 1993 Jennings, Andrew; Higuchi, Hideyuki (1993): “A User Model Neural Network for a Personal News Service.”; In: *User Modeling and User-Adapted Interaction*, 3(1); pp. 1-25.

Kals & Lutters, 1998 Kals, H.J.J.; Lutters, D. (1998): “The role of Information management in intelligent manufacturing”; In: *Proceedings of the CIRP International Conference on Intelligent Computation in Manufacturing Engineering: ICME’98*; Capri; pp. 21-28.

KAON developers guide, 2004 FZI (Ed.) (2004): “Developer s Guide for KAON 1.2.7”, FZI Research Center for Information Technologies at the University of Karlsruhe (Germany), Research Group Knowledge Management (WIM) and Institute AIFB, University of Karlsruhe (Germany), Knowledge Management Group (WBS)

Kobsa & Pohl, 1995 Kobsa, Alfred; Pohl, Wolfgang (1995): “The User Modeling Shell System BGP-MS”; In: *User Modeling and User-Adapted Interaction, 4*; pp. 59-106

Kolb, 2001 Kolb, Robert (2001):“Entwicklung eines Prüfmodells für die Qualitätssicherung von Karosseriebauteilen”, diploma thesis; University of Stuttgart, Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen ISW and DaimlerChrysler Research & Technology, Ulm (Germany).

Koopman, 2004 Koopman, Hilko (2004): “Engineering Object Relations in Automotive Engineering”; diploma thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twente, Enschede (The Netherlands) and DaimlerChrysler Research & Technology, Ulm (Germany).

Krause et al., 1991a Krause, F-L.; Kramer, S.; Rieger, E.; (1991): “PDGL. A Language for Efficient Feature-Based Product Gestaltung”; In: *Annals of the CIRP, 40/1*; pp. 135-138.

Krause et al., 1991b Krause, F-L.; Ulbrich, A.; Vosgerau, F. H. (1991): “Feature Based Approach for the Integration of Design and Process Planning Systems”; IFIP; J. P. A. M. W. J. Turner (Ed.), pp. 285-297.

Krause et al., 2001 Krause, F.-L.; Baumann, R.; Jansen, H.; Kaufmann, U. (2001): “Innovationspotentiale in der Produktentstehung – durchgängig digitale Prozesse mittels integrierter Virtueller Produktentstehung (iViP)”; In: *Industrie Management 3/2001*.

Krause et al., 2002 Krause, F.-L.; Tang. T.; Ahle, U. (2002): “Leitprojekt integrierte Virtuelle Produktentstehung – Abschlußbericht“; Fraunhofer IRB Verlag, Stuttgart

Krause et al., 2003 Krause, F.-L.; Baumann, R.; Kaufmann, U.; Kühn, T.; Leemhuis, H.; Ragan, Z.; Swoboda, F. (2003): “Computer Aided Conceptual Design”; In: *Proceedings of the 36th CIRP-International Seminar on Manufacturing Systems*, 03-05 June 2003, Saarbruecken (Germany)

Kruse et al., 1991 Kruse, R.; Schwecke, E.; Heinsohn, J. (1991): “Uncertainty and Vagueness in Knowledge bases Systems”; Springer-Verlag, Heidelberg

Kümmeth, 2004 Kümmeth, Sven (2004): “Einführung in die High Level Architecture (HLA)”; seminar thesis; Universität der Bundeswehr München, Fakultät für Informatik, Institut für Technische Informatik, Neubiberg (Germany)

Layer et al., 2001 Layer, A.; Haasis, S.; Van Houten, F. J. A. M. (2001): “Feature-based, design-concurrent cost calculation using case-based reasoning”, In: *Proceedings of ASME 2001/DETC*, paper number DETC01/DFM-21176, CD-ROM.

Layer, 2003 Layer, Alexander (2003): “Case-based Cost Estimation – A Building Block for Product Cost Management and Design-for-X“; doctoral thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twente, Enschede (The Netherlands); ISBN 90-365-1961-6

Lecluse, 1999 Lecluse (1999): “A design support system for three dimensional components and assemblies”; doctoral thesis; Lehven (Belgium).

Leemhuis et al., 2002 Leemhuis, Helen; Baumann, Richard; Kaufmann, Uwe; Swoboda, Frieder; Kühn, Thomas; Ragan, Zbigniew (2002): “Function Oriented Product

Modelling Based on Feature Technology and Integrated Constraint Management“; Product Data Technology Europe 2002, 11th Symposium, Centro Ricerche Fiat, Turin, Italy, 7th – 9th May.

Leemhuis, 2004 Leemhuis, Helen (2004): "Funktionsgetriebene Konstruktion als Grundlage verbesserter Produktentwicklung"; doctoral thesis; Fakultät V für Verkehrs und Maschinensysteme, Technische Universität Berlin.

Lenat, 1998 Lenat, Doug (1998), “The Dimensions of Context Space”; Cycorp, Austin, TX (U.S.A.); document available online at URLs http://www.cyc.com/publications.html or http://www.casbah.org/resources/cycContextSpace.shtml

Lutters et al., 1998 Lutters, D.; Streppel, A.H.; Kals, H.J.J. (1998): “Product information structure as the basis for the control of design and engineering processes”; In: *CIRP Journal of Manufacturing Systems*; Vol. 27, No. 2; Faculty press international; ISSN 0176-3377; pp. 199-204

Lutters & Kals, 1999 Lutters, D.; Kals, H.J.J. (1999): “Control of design and manufacturing processes based on information content”; CIRP

Lutters et al., 1999 Lutters, D.; Brinke, E. ten; Wijnker, T.C.; Kals, H.J.J. (1999): “Design and manufacturing processes based on Information Management”; In: *Proceedings of the 15th International Conference on Production Research*, Limerick; ISBN 1-874653-56-9; pp. 1375-1378.

Lutters, 2001 Lutters, D. (2001): “Manufacturing integration based on information management”; doctoral thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twente, Enschede (The Netherlands).

Maedche et al., 2001 Maedche, Alexander; Motik, Boris; Silva, Nuno; Volz, Raphael (2001): “MAFRA – A MApping FRAmework for Distributed Ontologies”; Forschungszentrum Informatik at the University of Karlsruhe (Germany).

Martin, 1996 Martin, Robert C. (1996): “Applying the Booch Method“, In: *OOP*.

Mellens, 2005 Mellens, Jannes (to be published 2005): “Design and Developement of a Prototype for Universal Linking of Engineering Objects”; diploma thesis; Faculty of Computer Science, University of Twente, Enschede (The Netherlands) and DaimlerChrysler Research & Technology, Ulm (Germany).

Mentink et al., 2002 Mentink, R.J.; Wijnker, T.C.; Lutters, D.; Kals, H.J.J (2002): “Supporting Manufacturing Environments; Utilizing integrated information in process support and control”; In: *Proceedings of the 4th annual University Synergy Program international conference (USP)*; Texas Tech University, Lubbock, Texas, USA; 14 pages on CD-ROM.

Mentink, 2004 Mentink, R. (2004): “Process Management in Design & Engineering; Applying dynamic process modeling based on evolving information content”; doctoral thesis; Department of Engineering, Laboratory for Design, Production and Management, University of Twente, Enschede (The Netherlands).

Miller et al., 1993 Miller, G. A.; Beckwith, R.; Fellbaum, C.; Gross, D.; Miller, K. (1993): “Introduction to WordNet: An On-line Lexical Database”; In: *Five Papers on WordNet.*

Narby, 2003 Narby, Eric (2003): “Express2UMEO“, report about practical internship; DaimlerChrysler Research & Technology, Ulm (Germany).

Nell, 1998 Nell, James G. (1998): “ISO TC 184 SC5 WG1 – Organization, Strategies, and Philosophy”; ISO TC 184 SC5 WG1 Modeling and Architecture; Document available online at URL http://www.cit.gu.edu.au/~bernus/taskforce/Meetings/brisbane98/technicalprogramme/pos-ppr/jimnell.html.

Nell, 2001 Nell, James G. (2001): “STEP on a Page”; Document available online at URL *http://www.nist.gov/sc5/soap*.

Nozick, 1990 Nozick, Robert (1990): “The examined Life“; Simon + Schuster, UK; ISBN 0-671-72501-7.

Oberle, 2004 Oberle, Daniel; Eberhart, Andreas; Staab, Steffen; Volz, Raphael (2004): “Developing and Managing Software Components in an Ontology-Based Application Server”; In: *Middleware 2004*; pp. 459-477.

Okeke, 2002 Okeke, Obumneme Nnamdi (2002): “Generalisierte Beschreibung feature-basierter Mess- und Auswertestrategien“, diploma thesis; University of Stuttgart, Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen ISW and DaimlerChrysler Research & Technology, Ulm (Germany).

OMG CADservices, 2003 OMG ManTIS workgroup “CAD Services V1.2 Revision Task Force” (Ed.) (2003): “CAD Services V1.0 Revised Submission”; OMG Document mfg/2001-06-03; document available online at URL http://mantis.omg.org/mfgcadv1-2rtf.htm

OMG MDAex, 2005 OMG (Ed.) (2005): “Model Driven Architecture, Executive Overview – The Architecture of Choice for a Changing World”, online available at URL *http://www.omg.org/mda/executive_overview.htm*.

OMG PDMenabler, 2000 OMG (Ed.) (2000): “STEP and OMG Product Data Management Specifications – A Guide for Decision Makers”; OMG Document mfg/99-10-04; PDES, Inc. Document MG001.04.00; document available online (amongst others) at URL http://mantis.omg.org/mfgppepdm.htm#PDMRTFV1.4.

OMG PLMservices, 2004 OMG (Ed.) (2004): “PLM Services Version 1.0”; OMG standard; document mantis/04-04-01 (Product Lifecycle Management Revised Submission); document available online at URL http://www.omg.org/cgi-bin/doc?mantis/2004-04-01.

Ontolingua, 1997 Standford University (Ed.) (1997): “Ontolingua Tutorial”; document available online at URL http://www-ksl-svc.stanford.edu:5915/doc/frame-editor/guided-tour/index.html.

Ostermayer, 2001 Ostermayer, Rainer (2001): “Pragmatisch-situative Wissensrepräsentation – ein Baustein für das Wissensmanagement“; doctoral thesis; University of

Karlsruhe, Fakultät für Maschinenbau, Shaker-Verlag.

Ourari, 2001 Ourari, Kaies (2001): "Erstellung einer Methodik für den Einsatz von ‚Combined Features' im Concurrent Engineering Prozeß bei der Entwicklung von Roh- und Fertigteilen für Aggregate"; diploma thesis, University of Stuttgart, Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen, ISW and DaimlerChrysler Research & Technology, Ulm (Germany).

Pepper, 2002 Pepper, S. (2002): "The TAO of Topic Maps – Finding the Way in the Age of Infoglut"; ontopia, document available online at URL http://www.ontopia.net/topicmaps/materials/tao.html.

Pepper & Garshol, 2002 Pepper, Steve; Garshol, Lars Marius (2002): "The XML Papers: Lessons on Applying Topic Maps"; In: *Proceedings of XML 2002*; deepX, pdf document available online at URL http://www.idealliance.org/papers/xml02/dx_xml02/papers/04-03-01/04-03-01.pdf or as html at URL http://www.ontopia.net/topicmaps/materials/xmlconf.html.

Petkova, 2001 Petkova, Maja (2001): "Automation der CAD/CAQ-Prozeßkette zur Qualitätssicherung von Karosseriebauteilen auf der Basis von Features"; diploma thesis; Technical University of Sofia, and DaimlerChrysler, Sindelfingen (Germany).

Ping, 2002 Ping, Ge (2002): "An Axiomatic Approach for 'Target Cascading' of Parametric Design of Engineering Systems"; In: *CIRP Annals 2002*

PROSTEP, 2003 ProSTEP AG (2003): "Product Lifecycle Management Service RFP – Overview on the DaimlerChrysler Initial Submission http://www.omg.org/cgi-bin/doc?mantis/03-05-03"; ProSTEP AG, Darmstadt (Germany); document available online at the URL specified in the title.

Protege, 2004 Standford University (Ed.) (2004): "What is Protégé-2000?"; tutorial; document available online at URL http://protege.stanford.edu/whatis.html

Rahm & Bernstein, 2001 Rahm, E.; Bernstein, P. (2001): "A survey of approaches to automatic schema matching."; In: *VLDB Journal*, 10(4); pp. 334-350.

Raymond, 2000 Raymond, Eric (2000): "Why Python?"; In: Linux Journal; document available online at URL http://www.linuxjournal.com/article.php?sid=3882.

Rich, 1979 Rich, Elaine (1979): "User Modeling via Stereotypes"; In: *Cognitive Science*, 3(3); pp. 329-354.

Rich, 1979b Rich, E. (1979): "Building and exploiting user models"; doctoral thesis; Carnegie Mellon University, Pittsburgh (U.S.A.).

Rossum, 1997 Van Rossum, Guido (1997): "Comparing Python to Other Languages"; documentation; document available online at URL http://www.python.org/doc/essays/comparisons.html

Rumbaugh et al, 1991 Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorenson, W. (1991):

“Object-Oriented Modelling and Design”; Prentice-Hall, Englewood Cliffs.

Salomons et al., 1993 Salomons, O. W.; van Slooten, F.; van Houten, F.J.A.M.; Kals, H.J.J. (1993): “FROOM-A Demonstration Session”; In: *KNOWHSEM 1993*; pp. 279-287.

Salomons, 1995 Salomons, Otto Willem (1995): “Computer Support in the Design of Mechanical Products – Constraint specification and satisfaction in feature-based design for manufacturing“; doctoral thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twente, Enschede (The Netherlands), ISBN 90-9007877-0.

Schulze et al, 1999 Schulze, H.; Haasis, S; Witt, H. (1999): “Gestufte Integration vorhandener Erfahrung in die Feature-basierte Produktionsarbeit“; In: *Tagungsband des 11. Züricher Symposiums für Arbeitspsychologie*, (Band 11); Symposium für Arbeitspsychologie, Zürich (Eds.); *[Step by step integration of existing experience into feature based work]*

Schumann & Müller, 2004 Schuhmann, Heidrun; Müller, Wolfgang (2004): “Informationsvisualisierung: Methoden und Perspektiven”; In: *Information Technology it*, 3/2004, Oldenbourg Verlag; pp. 135, 141

Schwarz et al., 2002 Schwarz, Johann; Zimmermann, Johann U.; Weimer, Henrik; Frank, Dietmar; Haasis, Siegmar (2002): “The Use of e-Business Techniques in Feature-Based Automotive Product Development“; In: *Proceedings of the ICE2002*.

Shah, 1988 Shah, J. J. (1988): “Feature transformations between application-specific feature spaces”; In: *Computer-Aided Engineering Journal*, 5/6; pp. 247-255.

Shah et al., 1990 Shah, J. J.; Rogers, M. T.; Sreevalsan, P.; Hsiao, D. W.; Mathew, A.; Bhatnagar, A.; Liou, B. B.; Miller, D. W. (1990) “The A.S.U. features test bed. An overview”, In: *Proceedings of the 1990 ASME.*

Shah et al., 1993 Shah, J. J.; Hsiao, D.; Leonard, J. (1993): “A Systematic Approach for Design-Manufacturing Feature Mapping”; In: *Geometric Modeling for Product Realization. IFIP 1993*; M. J. W. A. M. J. P. P.R. Willson (Ed.); pp. 205-221.

Sowa, 1984 Sowa, John F. (1984): “Conceptual structures, information processing in mind and machine”; Addison-Wesley Publishing Company Inc., Reading; ISBN 0201144727.

Sowa, 1992 Sowa, J.F. (1992): “Conceptual graphs summary”; In: *Conceptual structures, current research and practice*; Nagle, T.E.; Nagle, J.A.; Gerholz, L.L.; Eklund, P.W. (Eds.); Ellis Horwood Ltd., Chichester; ISBN 0131758780; pp. 3-52.

Sowa, 2000 Sowa, J.F. (2000): “Knowledge Representation: Logical, Philosophical, and Computational Foundations”; Brooks Cole Publishing Co., Pacific Grove, CA (U.S.A.); ISBN 0-534-94965-7.

Srikantappa & Crawford, 1992 Srikantappa, A. B.; Crawford, R. H. (1992): “Intermediate Geometric and interfeature relationships for automatic group technology part coding”; In: *Proceedings of ASME 1992*; pp. 245-251.

Steiss, 2003 Steiss, Ekkehard (2003): “Funktionsintegration bei CAx-Werkzeugen“; diploma thesis; Faculty of Computer Science, University of Stuttgart (Germany), and DaimlerChrysler Research & Technology, Ulm (Germany)

Suh, 1990 Suh, N. P. (1990): “The Principles of Design”; Oxford University Press, New York (U.S.A.).

Suh, 1999 Suh, N.-P. (1999): “Applications of axiomatic design”; In: *Integration of process knowledge into design support systems*; Kals, H.J.J.; van Houten, F.J.A.M. (Eds.); Kluwer Academic Publishers; Dordrecht; ISBN 0792356551; pp. 1-46

Suh, 2001 Suh, N. P. (2001): “Advanced Axiomatic Approach and Applications”, Oxford University Press, New York (U.S.A.).

Suh & Wozny, 1998 Suh, Y. S.; Wozny, M. J. (1998): “Interactive Feature Extraction for a Form Feature Mapping System”; Rensselaer Polytechnic Institute Troy, New York (U.S.A.).

Suh, 1995 Suh, Y. S. (1995): “A Feature-Conversion CAD System for the Concurrent Engineering Environment”; doctoral thesis; Rensselaer Polytechnic Institute, Troy, New York (U.S.A.).

Thome, 2003 Thome, Mario (2003): “Feature-basierte Prüfplanung und -durchführung am Beispiel von Karosserieteilen mit den Systemen CATIA V5 und DELMIA”; diploma thesis; University of Saarbrücken, Lehrstuhl für Werkstofftechnologie/Präzisformgebung (Germany), and DaimlerChrysler AG Sindelfingen (Germany).

Tichem & Storm, 1996 Tichem, Marcel; Storm, Ton (1996): “Issues in Product Structuring”; In: *Proceedings of the 2^nd^ WDK Workshop on Product Structuring*; Delft University of Technology, Delft (The Netherlands).

Tönshoff et al., 1997 Tönshoff, H.K.; M. Ehrmann, G. Zahn (1997): “Technische Elemente zur Integration von CAD und CAPP.“; In: *VDI-Berichte 1322 – Features verbessern die Produktentwicklung/Integration von Prozeßketten*; VDI-Verlag Düsseldorf; ISBN 3-18-091322-3; pp. 179-194.

Van den Elst, 2002 Van den Elst, Tom (2002): “Development of a Software Prototype for Universal Linking of Engineering Objects (ULEO)“; Diploma thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twente, Enschede (The Netherlands) and DaimlerChrysler Research & Technology, Ulm (Germany).

Van Houten, 1991 Van Houten, Fred J. A. M. (1991): “PART: a computer-aided process planning system”; doctoral thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twente, Enschede (The Netherlands).

Van Vliet & Van Luttervelt, 1999 van Vliet, J.W.; van Luttervelt, C.A.; Kals, H.J.J. (1999): “STATE-OF-THE-ART REPORT ON DESIGN FOR MANUFACTURING“; DETC99/DFM-8970; In: *Proceedings of the 1999 ASME Design Engineering Technical Conferences*; September 12–15, 1999, Las Vegas, Nevada (U.S.A.)

VDI-2218, 1999 VDI (Ed.) (1999): “VDI-Richtlinie 2218: Feature-Technologie“, Stand v. 2.11.1998; new release from 1999; VDI-Verlag, Düsseldorf

W3C XML, 2000 W3C (Ed.) (2000): “Extensible Markup Language (XML) 1.0 (Second Edition)”; W3C Recommendation; Document available online at URL http://www.w3.org/TR/2000/REC-xml-20001006.

W3C DAML+OIL, 2001 W3C (Ed.) (2001): “DAML+OIL (March 2001) Reference Description”; W3C Note 18; document available online at URL http://www.w3.org/TR/daml+oil-reference.

W3C XMLschema, 2001 W3C (Ed.) (2001): “XML Schema”; W3C Recommendation; Document available online at URL *http://www.w3.org/2001/XMLSchema*.

W3C OWL, 2004 W3C (Ed.) (2004): “OWL Web Ontology Language Overview”, W3C Recommendation; document available online at URL http://www.w3.org/TR/2004/REC-owl-features-20040210/.

W3C OWLb, 2004 W3C (Ed.) (2004): “OWL Web Ontology Language Guide”, W3C Recommendation; Document available online at URL http://www.w3.org/TR/owl-guide/.

W3C RDFprimer, 2004 W3C (Ed.) (2004): “RDF Primer”, W3C Recommendation; Document available online at URL http://mirrors.webthing.com/view=Asis/www.w3.org/TR/2004/REC-rdf-primer-20040210/.

W3C press, 2005 W3C (Ed.) (2005): “World-Wide Web Consortium Issues RDF and OWL Recommendations”; W3C press release; Document available online at URL http://www.w3.org/2004/01/sws-pressrelease.html.en.

Welty et al., 2001 Welty, Chr.; Smith, B. (2001): “Formal Ontology in Information Systems”, In: *Proceedings of the 2nd FOIS conference*, Oct. 2001; ACM Press, New York (U.S.A.).

Wijnker et al., 2000 Wijnker, T.C.; Lutters, D.; Kals, H.J.J. (2000): “The use of workbenches based on Information Management”; In: *University synergy program: round table*; Haifa; on CD-ROM.

Wijnker, 2003 Wijnker, T.C. (2003): “Integration of Information in Manufacturing Systems”; doctoral thesis; Dept. of Engineering, Laboratory for Design, Production and Management, University of Twente, Enschede (The Netherlands); ISBN: 90-365-1867-9.

Wong & Leung, 2000 Wong, T. N.; Leung, C. B. (2000): “An object-oriented neutral feature model for feature mapping”; In: *International Journal of Production Research*, 38; pp. 3573-3601.

XTMspec, 2001 TopicMaps.org (Eds.) (2001): “XML Topic Maps (XTM) 1.0 – TopicMaps.Org Specification”; Document available online at URL http://www.topicmaps.org/xtm/1.0/.

Zadeh, 1979 Zadeh, Lotfi Asker (1979): “A theory of approximate reasoning.”; In:

Machine Intelligence, 9; pp. 149-194.

Zimmermann, 1994 Zimmermann, Johann (1994): “Hybride Wissensrepräsentation in BGP-MS. Integration der Wissensverarbeitung von SB-ONE und OTTER”; diploma thesis; University of Konstanz (Germany); Bericht 62-94 (WIS-Memo 12); abstract document available online at URL http://www.inf-wiss.uni-konstanz.de/FG/94da.html.

Zimmermann et al., 2002a Zimmermann, Johann U.; Haasis, Siegmar; van Houten, Fred J. A. M. (2002): “ULEO – Universal Linking of Engineering Objects”, In: *CIRP Annals*, Vol. 51/1 – 2002.

Zimmermann et al., 2002b Zimmermann, Johann U.; Haasis, Siegmar; Van Houten, Fred J.A.M. (2002): “Applying universal linking of engineering objects on the automotive industry – practical aspects, benefits, and prototypes”, In: *Proceedings of the 2002 ASME DETC conference*.

Zimmermann et al., 2004 Zimmermann, Johann U.; Karthe, Thomas; Biedenbach, Hans-Martin (2004): “Integration von Qualitätssicherungsanwendungen in der Automobilindustrie“; 3 parts; In: *CAD-CAM Report* No. 11+12/2004, and No. 1/2005, Dressler Verlag, pp. 48 ff / 36ff / 14ff.

Chapter 21 Coherent Glossary of Important Terms

This appendix introduces some terms used throughout the thesis and explains their meaning as presupposed here. See also the listed glossary below.

A **datum** (pl. **data**) is a representation of a complete or incomplete proposition (in the broad sense). **Information** is some kind of data plus a context-specific **meaning** (semantics) – to simplify the detailed definitions, the terms "semantics" and "meaning" are used as synonyms in this thesis. **Context** is a kind of information embedding other information. **Semantics** is a kind of information. This implies a recursive character of information. A simple context is located inside a computer – a complex one inside a human's mind. Semantics arises from the **interpretation** of data in an embedding context. **Knowledge** is information within the (subjective) context of a conscious being. It therefore only exists within some conscious being's head. – These definitions of the terms "data" and "information" coincide, for instance, with those used in *[Van Vliet & Van Luttervelt, 1999]*. The understanding of knowledge deviates, however.

Atomary pieces of information with their own meaning (such as concepts and relations) within the information system are also called **informational entities** (**IEs**).

Objects are independent entities in the real world (the domain), e.g., cars, motors, cylinder heads, holes, tolerances, and costs, or that exist in a virtual sense inside software representations. Objects have properties. Properties can be divided into two groups with respect to their ability to characterize the object they are associated to. However, there is no rule where to draw the line: it depends on the domain and the requirements placed on the computer model: **Object-defining properties** (primary-, direct-, internal properties) are encapsulated within the object in terms of attributes and methods and are independent from other objects. **External properties** (secondary-, indirect properties) consist of relations to other objects.

Additionally, it is possible to differentiate **static properties** as described above from **procedural** ones. The procedural properties (termed methods in the object-oriented paradigm OO) can be directly or indirectly executed by computers in order to perform certain actions related to one object. It is a more philosophical question to decide whether procedural information is also a real object property or merely part of its usage.

Objects can be **classified** according to their set of property types. Thus, specific objects may be **instances** of a class, and consequently, there are abstract and instantiated objects (classes and instances). Classes are a means of grouping individual objects of the domain to simplify their handling. They are thus one step higher up in abstraction than instances. Classes describe the properties that are relevant and identifying for all their instances (internal properties) as well as, optionally, the values that these properties may assume. If the process of abstraction is also applied to classes, a hierarchy of classes, a so-called **taxonomy**, originates. In a taxonomy, more abstract parent classes subsume less abstract child classes, and the latter inherit internal properties from their parents.

✍ A similar OO concept is that of interfaces (in the following referred to as **OO interfaces**). Similar to classes, they encapsulate primary properties. They typically do not represent objects, however, but views on objects, in that they cover the properties needed for supplying an object (class) with certain co-natural characteristics. However, these characteristics are generally of relevance for more than one class. A class can make use of them by "implementing" an interface. The term "OO interface" subsumes EO interfaces and EOR interfaces.

The representation of a real-world relationship within a computer model or data structure is termed a **relation.** Depending on the requirements to be met by information processing, **relationships** may be regarded as an independent type of entity within the real-world domain and within computer models. In this case, they can be considered objects, and, consequently, the notion of an object subsumes both concepts and relations. This approach is also adopted in this thesis. This implies that both may carry attributes and methods. Consequently, relations may also be classified according to their properties (attributes and methods). The relation classes (relation types, abstract relations) may be represented in an own model. Hence, relations correlate not only abstract or instantiated concepts but also other abstract or instantiated relations. The **arity** of relations is a further key concept: a relation connecting *n* objects is termed an *n-ary* or *n-adic*[*] or *n-way* relation. If *n* equals 2, the relation is termed *binary* or *dyadic*.

The presence of **associativity** between informational entities means that relations are maintained and deployed between them. Uni-directional associativity means tracking relationships in a single direction. Multi-directional associativity means tracking relationships any way the logical relationships foresee. The kind of associativity, however, does not say anything about the complexity of the information and the quality of its deployment.

Collections of abstract concepts and relations are also called **abstract** or **background information**, whereas collections of instantiated ones are also called **specific** or **instance information**. Instances of concepts are also called **individuals**.

✍ **Background information** is abstract, general, class-like, type information describing the nature of information as handled in product development, for example, and may also subsume company know-how. **Specific information** describes individual instances. Background and specific information are complementary. While specific and background information have static character, **control information** is dynamic and influences the usage of the static information and the cooperation of applications (workflow).

✍ **Knowledge** comprises abstract and specific information. However, within this thesis, knowledge is almost exclusively used to designate such kind of abstract information. Deviating from the given definition of the term knowledge, and for reasons of consistency with the common, more liberal definition of knowledge, the term knowledge is used synonymously for **complex information** and for **high-level information**.

Meta-information documents information about information, for instance, the properties and meaning of relation types, context information, and the semantics of EO classes.

A source of information (sender) assigns a certain **meaning** (**semantics**) to informational entities that reflects its own **view** on it. For the semantically correct interpretation of the informational entities' contents, it is important that the information destination knows the semantics defined by the sender. This implies that semantics is **context-dependent** by its nature. The description of the meaning of some piece of information does not necessarily have to be located and represented separated from the user information. If separated, however, such a model is referred to as an **ontology** here (see below). In either case, such information is called **self-contained** that is equipped with meta-information describing its semantics (**self-contained information**, **SCI**).

The purpose of an **ontology (**see *[Welty et al., 2001]* for an overview of ontology-related issues) is to describe the meaning of some piece of information that is not part of the ontology. Ontologies in computer science are used to describe the **meaning of concepts** employed within some piece of specific information. Based on an ontology, computers and, in the end, humans are to be enabled to interpret incoming data in a way that is similar to the interpretation that the respective data source applied. This means that they should be enabled to restore information as accurately as possible after the information has bees transferred between information processors. Thus, ontologies are a means of enhancing the transferability and the shared usage of information within or between domains

Footnotes

[*] As used by Sowa (see *[Sowa, 2000]*) for conceptual relations

✍ **Ontologies**, in the information-technological sense, typically describe some kind of semantic net represented using a formalism with a semantic specification. In several cases of existing ontology-based IT approaches, this formalism is enriched by the option to specify logical statements in order to enhance its expressive power. Current ontology definition languages typically try to achieve this by describing the **properties** of a set **of** abstract **concepts** that are formalized and generalized classifications of some entities occurring in the respective application domain and considered relevant for the targeted information processing. These concepts are represented inside a model, namely the ontology. In the most general view, such concept properties are represented in terms of relationships to other concepts in the model. Some approaches are optimized according to certain aspects in that they discern the concept-defining types of a concept’s relations from its external ones that are defining its relationships to other concepts.

Informational entities may be described within a **model**, for example, an **information model**. An **information model** can be an **ontology**, but not every ontology is an information model.

If such an information model – or information as such – exists not only inside a human’s brain but also inside a computer, it has to be technically represented by means of a **representation formalism**. For instance, a textual language (programming language, XML, etc.) can be used, or any other set of textual or binary representation structures that are typically consisting of many encapsulated entities that are related to each other in some way (by pointers or keys, for instance). These entities might be visualized in any specific way – for instance by graphical symbols inside a diagram or in the form of tables or tree structures. Classes and instances of real-world objects may be represented in a wide variety of ways within computer-based models. Nevertheless it is important to keep in mind that most representation formalisms still try to describe (part of the) **objects** (including the relations) existing in the real-world domain:

~ ***Symbolic*** **representation formalisms** assign certain symbols to certain objects of the domain in order to represent these entities inside the computer-based model. ***Sub-symbolic*** approaches employ a changing representation for given entities in the domain. Consequently, they are harder for humans to read and understand.

~ Furthermore, the properties of a given IE may be represented within a single, **strictly encapsulated** bulk of information (such as an object-oriented class definition) or may be distributed over several elements of the model (as is possible in logical representation formalisms).

Engineering objects (**EOs**) are objects that represent concepts relevant for product engineering, for example, features, parts, assemblies, surfaces, and tolerances. Adopting the terminology used in the OO paradigm, abstract concepts are also termed **engineering object classes**, while instantiated concepts are also denoted as **engineering object instances** (**EOIs**). Analogously, relations are termed **engineering object relations**. **Specific information** comprises EOIs and instantiated relations, the **EORIs**[*]. EO(R)Is are used to construct **product, process and resource (PPR) models** from the viewpoint of an individual application: they represent exactly one informational entity in the considered part of the real world, the so-called **domain**.

☞ In this thesis, the terms “**product model**” and “**PPR model**” are widely exchangeable, as the latter subsumes the former and as the propositions made on product models are generally true also for PPR models. However, as the term “product model” is still broader introduced in the field of product engineering, it eases referencing of and comparison to existing approaches.

Features are considered to be engineering objects that are used to model a product, process or resource from the perspective of a certain step during product development. **User-defined features**

Footnotes

[*] More details such as the various forms of incarnations (EOR type, EORM) are explained in the concept part of this thesis.

(**UDFs**) are features which are defined and implemented by company-specific end-users or standardization committees, without any means of programming and rebuilding the CAx system. They usually represent intellectual property of a company.

Products may consist of one or more **assemblies**, which, in turn, may consist of one or more **parts**. Thus, features usually represent constituents of parts and contain some kind of geometry description (faces, edges, solids, etc.). If this is the case, features are "smaller" than parts but "bigger" than solids and faces. However, depending on the application they are designed for, features may carry geometric and/or non-geometric information.

EO constellations (**EOCs**) or **combined features** (**CFs**) are groups of EOs meant to be instantiated in a single user operation. They are represented by special EO relations.

The field of **IT for engineering** subsumes information-technological means utilized or appropriate for supporting product engineering processes. Each software application along the product development process chain (**process step application – ProSAp**) has to fulfill specific tasks using specialized technical (human) knowledge and modus operandi for problem-solving.

The nature of the term **informational**[*] **integration** as used during the following investigations can be informally defined as the "multi-directional flow of complex information plus multi-directional associativity inside a global information space", which means that informational entities are managed by certain applications that allow other applications or users to access the informational entities in their original location (**information sharing**) or to create copies of them (**information replication**), to store the copies, for example, in files, and to hand them over (**information exchange**) to interested applications. **Information flow** subsumes sharing and replicating information. The flow direction of information is independent of its complexity. A **global information space** (**GIS**) overspans a set of running software applications and users that are able to share and exchange information according to their needs without being significantly hindered by technical limitations. The information to be shared is also called *user information*, emphasizing the fact that it does not comprise portions of infrastructural information, for example, from the transportation layer. **Informational base types** or **basic information types** are principle kinds of informational entities that occur in the GIS such as abstract and instantiated concepts and relations and, more specifically, EO class, EO instance, EOR type, etc. The **GIS information structure** (or shortly GIS structure) comprises accessible information pools/sources in the GIS, whose IEs adhere to specific informational base types.

Feature linking[†] is informally defined here as feature mapping together with the generation and maintenance of persistent links between the mapped feature instances. The term is also used for EO linking.

✎ Note about the *terminology*: This research started off as an investigation into approaches for feature mapping, proceeded to feature linking, and has now reached the concept of *Universal Linking of Engineering Objects*. As it is not feasible to change names given for the approaches too often over time, the author decided to keep the methods' names stated below, even if the notion of *features* has been replaced by that of *engineering objects* in the meantime. So, ULEO is partly still termed feature linking. It is important to point out, though that every appearance of features is also valid for EOs.)

Footnotes

[*] In the sense of "on the information level"

[†] The notion of feature linking has previously been employed by Bronsvoort *[Bronsvoort et al., 2001]* with a different meaning: feature links consist of a set of constraints that are used to insure consistency between certain form features. Form features are commonly known as features containing geometry only).

In the context of this work, **flexibility** of software is the degree to which its functionality may be influenced without changing its source code. Flexibility and – as the software must not contain too much overhead – **modularity** are, among other factors, prerequisites for **scalability,** i.e., for the option to adapt software to the practical needs within a given domain in a company.

A **user** or end-user is a person (in general, an engineer) who utilizes a software application in order to fulfill a task as part of the product development process, for example, in concept design, detail design, manufacturing planning, or assembly planning.

Chapter 22 Listed Glossary of Terms and Shortcuts

This appendix lists concise explanations. See also the preceding appendix for a coherent introduction.

Several definitions are taken from the internet web site WordIQ.com[*]

Term	Explanation	Synonyms
AC	AutoCreate relation, a special kind of EOR	
AF	Assembly feature	
API	Application programming interface	
ASME	Acronym for American Society of Mechanical Engineers	
Assembly feature	Kind of engineering object that concentrates all the information relevant for a certain assembly situation given by a set of n individual EOs and respective destination product parts	
BNC	Binary numeric control	CNC
Body-in-white	A car’s inner, load-bearing sheet metal construction that is bare of outer design-forming elements	
CAA	Component Application Architecture; application programming environment by Dassault Systèmes; consists of C++ libraries and a development environment	
CAD	Computer-aided design	
CAM	Computer-aided machining	
CAPP	Computer-aided production planning	
CATIA™	CAx system from Dassault Systèmes, France. Offers several so-called work-benches, all using very similar user interfaces, each of which is dedicated to fulfilling tasks of a certain step in product development	
CAx	Collective term subsuming all computer-aided steps within product development such as CAD, CAM, CAPP	
CIRP	International Institution for Production Engineering Research	
CMM	Coordinate measuring machine	
CNC	Computerized numerical control	BNC
Conceptual graphs	*John F. Sowa*'s **Conceptual Graphs** allow the graphical statement of *logic propositions* or *predicates*. Sowa credits the **Existential Graphs** of Charles Sanders Peirce for his conceptual graphs. *[wordIQ.com[†]]*	
Conceptual schema	A **conceptual schema** is a map of concepts and their relationships. An incomplete list of concepts is set out below: - Physical object (an *instance* of something)	

Footnotes

[*] See the URL *http://www.wordiq.com/de*

[†] See the URL *http://www.wordiq.com*

	- Abstraction (a *class* of something) An incomplete list of relationships is set out below: - A *is a* B - A *contains* B - A *requires* B The next step towards creating a database design, after a conceptual schema is produced, is a logical schema. *[wordIQ.com]*	
Csv file	Comma-/character-separated value file; text file containing line-oriented information. Each line has the same structure and the individual elements in that structure are separated by commas or other characters.	
DECC	Digital Engineering Competence Center; a software laboratory at DaimlerChrysler Research & Technology, Ulm, equipped with the technical infrastructure to embed software prototypes into the realistic environment of scenarios from the domain of product development.	
Description logics	Description logics are knowledge representation languages tailored for expressing knowledge about concepts and concept hierarchies. They are usually given a Tarski style declarative semantics, which allows them to be seen as sub-languages of predicate logic. They are considered an important formalism unifying and giving a logical basis to the well known traditions of frame-based systems, semantic networks and KL-ONE-like languages, object-oriented representations, semantic data models, and type systems. The basic building blocks are concepts, roles and individuals. Concepts describe the common properties of a collection of individuals and can be considered as unary predicates which are interpreted as sets of objects. Roles are interpreted as binary relations between objects. Each description logic defines also a number of language constructs (such as intersection, union, role quantification, etc.) that can be used to define new concepts and roles. The main reasoning tasks are classification and satisfiability, subsumption and instance checking. Subsumption represents the is-a relation. Classification is the computation of a concept hierarchy based on subsumption. A whole family of knowledge representation systems have been built using these languages and for most of them complexity results for the main reasoning tasks are known. Description logic systems have been used for building a variety of applications including conceptual modeling, information integration, query mechanisms, view maintenance, software management systems, planning systems, configuration systems, and natural language understanding. *[Patrick Lambrix[*]: Description Logics homepage[†]]*	DL
Design-for-X	Concept for widening engineers' informational scope. Design-for-X strives to support product designers with such information, allowing them to consider downstream processes' aspects during their design,	Design-to-X, DfX

Footnotes

[*] See the URL *http://www.ida.liu.se/labs/iislab/people/patla/*

[†] See the URL *http://www.ida.liu.se/labs/iislab/people/patla/DL/index.html*

	thus achieving a satisfactory design earlier with fewer feedback loops.	
Design-to-X		Design-for-X
DF	Design Feature	
Domain view	Sub-taxonomies and contexts within the IIM are a sort of views within the global information space. These are views on the engineering domains’ objects and relations, describing products, processes, resources and so on.	
Downstream process	Here: steps in the product development subsequent to detail design	
EDM	Engineering Data Management	
EO	Engineering object; objects representing concepts relevant for product engineering, e.g., features, parts, assemblies, surfaces, and tolerances	
EOC	Engineering object constellation, several EOx correlated via one EOR	
EOCL	EO constellation linking	
EOI	Engineering object instance	
EOR	Engineering object relation	
EORI	Engineering object relation instance	
EORM	Engineering object relation materialization	
EOx	Collective term for EO class, EOI, EOR type, EORM, EORI	
FCL	Feature constellation linking	
Feature	A kind of engineering object; *Features* are considered to be objects (concepts) used to model a product from the perspective of a certain step during product development. Products may consist of one or more assemblies which, in turn, may consist of one or more parts. Thus, features usually represent components of parts and contain some kind of geometry description (faces, edges, solids, etc.). If this is so, features are “smaller” than parts but “bigger” than solids and faces. However, depending on the application they are designed for, features may carry geometric and/or non-geometric information. Feature types should correspond to the concepts the users have in mind while performing their tasks. Features assume the role of building blocks for creating view-specific product models. design features = finish-part features = view of detail design	
Feature linking	Feature linking is informally defined here as feature mapping together with the generation and maintenance of persistent links between the mapped feature instances.	
Feature Mapping	A transformation between two feature sets A and B is performed by generating the new set of feature instances B from the given one A. In the general case there may be n-to-m relations between the features of A and B.	feature transformation, feature conversion
First order	**First-order predicate calculus** or **first-order logic** (**FOL**) is a	predicate

logic (FOL)	theory in *symbolic logic* that permits the formulation of *quantified* statements such as "there is at least one X such that..." or "for any X, it is the case that...", where X is an element of a set called the domain of discourse. A **first-order theory** is a theory that can be axiomatized as an extension of first-order logic by adding a recursive set of first-order sentences as axioms. First-order logic is distinguished from higher-order logic in that it does not allow statements such as "for every *property*, it is the case that..." or "there exists a *set* of objects such that...". Nevertheless, first-order logic is strong enough to formalize all of set theory and thus virtually all of mathematics. Its restriction to quantification over individuals makes it difficult to use for the purposes of topology, but it is the classical logical theory underlying mathematics. It is a stronger theory than sentential logic, but a weaker theory than arithmetic, set theory, or Second-order logic. *[wordIQ.com]*	calculus, FOL, predicate logic
FOL	first-order logic	
Frame	From the *1960s*, the *knowledge frame* or just *frame* has been used. A frame consists of slots which contain values; for instance, the frame for *house* might contain a *color* slot, *number of floors* slot, etc. Frames can behave something like object-oriented programming languages, with inheritance of features described by the "*is-a*" link. However, there has been no small amount of inconsistency in the usage of the "is-a" link: Ronald J. Brachman wrote a paper titled "What IS-A is and is not", wherein 29 different semantics were found in projects whose knowledge representation schemes involved an "is-a" link. Other links include the "*has-part*" link. Frame structures are well-suited for the representation of schematic knowledge and stereotypical cognitive patterns. The elements of such schematic patterns are weighted unequally, attributing higher weights to the more typical elements of a schema. A pattern is activated by certain expectations: If a person sees a big bird, he or she will classify it rather as a sea eagle than a golden eagle, given his or her "sea-scheme" is currently activated. Frames representations are more object-centered than semantic networks: All the facts and properties of a concept are located in one place – there is no need for costly search processes in the database. Frames suffer from the *frame problem* of knowledge linking. A *script* is a type of frame that describes what happens temporally; the usual example given is that of describing going to a restaurant. The steps include waiting to be seated, receiving a menu, ordering, etc. Frame problem: In artificial intelligence, the **frame problem** has a number of possible formulations. One of the most common is that it is the question of how to determine efficiently which things remain the same in a changing world. John McCarthy and Patrick J. Hayes introduced the term "frame problem" in their 1969 essay, *Some Philosophical Problems from the Standpoint of Artificial Intelligence*.*[wordIQ.com]*	

GACI	German Automotive CATIA Initiative; a syndicate of all German automotive manufacturers (except for Adam Opel AG)	
GEOR	Generative engineering object relation; GEORs are introduced to facilitate dynamic automation behavior (micro workflows) and thus scalable systems	
GIS	A global information space overspans a set of running software applications and users that are able to share and exchange information in a sophisticated way. All kinds of relevant information are available to all GIS participants at the time they are needed. In a GIS, all the people and software systems share information according to their needs without being significantly hindered by technical limitations.	
Grammar	According to the structuralist point of view, **grammar** is the study of the rules governing the use of a language. That set of rules is also called the **grammar** of the language, and each language has its own distinct grammar. Grammar is part of the general study of language called linguistics. The subfields of grammar are phonetics, phonology, morphology, syntax, and semantics. In traditional terms, grammar includes only morphology and syntax. Programming languages used for the purpose of computer programming (such as *Java*) have grammars, but do not resemble human languages very much. These are called formal grammars. In particular, they conform precisely to a grammar generated by a pushdown automaton with arbitrarily complex commands. They usually lack questions, exclamations, simile, metaphor and other features of human languages. *[wordIQ.com]*	
GUI	Graphical user interface	
I++	The *Inspection-plusplus (I++) Workgroup* (say: »I plus plus«) is a workgroup of German and Swedish automotive OEMs[*] and inspection equipment manufacturers. It consists of experts in the fields of product quality assurance and information technology and is working on commitments that are to enable or push the cooperation of software applications along the CAD/CAQ process chain. The author takes part in this effort. The ultimate motivation is to obtain or offer, respectively, a variety of compatible software tools for quality assurance.	
IAC, IAS	Identification and addressing concept / schema; a data structure encapsulating all kinds of information necessary for identification and locating an IE. The term IAC stands for the underlying methodology of identification and addressing.	
ID	Identity	id
IE	informational entity; atomary piece of information within the	

Footnotes

[*] Such as BMW, DaimlerChrysler, Volkswagen/Audi, and Volvo

	information system (such as a concept or a relation with own meaning); informational entity = entity in a computer model	
IE view	informational entities' view; a user view on the informational entities in the global information space	
IEOR	Informational EOR; IEORs represent background information to be used by ProSAps (e.g., ontological information and automation knowledge).	
IIM	Integrated information model	
Information modeling	The process of creating a model of part of the real world within an information system and its representation by means of an information representation formalism. The model represents informational entities by means of representational elements (also called representation elements).	
Information representation formalism	A formalism to represent information within an information system	representation formalism
Information system	The human mind, computer	information processor, IT system
Informational entity		IE
Instance (information)	EO instances + EOR instances in ULEO GIS	instance-level information, specific information, facts, concrete information, distinct information
ISO	International Organization for Standardization	
ISRL	Inspection Strategy Representation Language	
IT	Information technology	
J2EE	Java 2 Platform, Enterprise Edition, defines the standard for developing component-based multitier enterprise applications *[Sun*]*	
Knowledge	***Knowledge*** is the awareness and understanding of facts, truths or information gained in the form of experience or learning. Knowledge is an appreciation of the possession of interconnected details which, in isolation, are of lesser value. Knowledge is a term with many meanings depending on context, but is (as a rule) closely related to such concepts as meaning, information, instruction, communication, representation, learning and mental stimulus. *[wordIQ.com]*	

Footnotes

[*] Sun microsystems, see the URL *http://www.sun.com/j2ee/*

Knowledge management	Knowledge management (KM) is the management of knowledge within organizations. *[wordIQ.com]*	
Knowledge representation	**Knowledge representation** is a central problem in arranging knowledge. It is needed for library classification and processing concepts in an information system. There are difficulties in the field of artificial intelligence. The problem consists of how to store and manipulate knowledge in an information system in a formal way so that it may be used by mechanisms to accomplish a given task. Examples of applications are expert systems, machine translation systems, computer-aided maintenance systems and information retrieval systems (including database front-ends). *[wordIQ.com]*	
KR language	Knowledge representation language	information representation language
MEOC	Manual EO constellation	
MF	Machining Feature	
MFC	Manual feature constellations	
MTRT	Meta-taxonomy of relation types	
NIST	National Institute of Standards and Technology, USA	
Object (in computer science)	An **object** is a unique concrete instance of an *abstract data type* (that is, a conceptual structure including both *data* and the *methods* to access it) whose identity is separate from that of other objects, although it can "communicate" with them via *messages*. *[wordIQ.com]*	
OEM	Original equipment manufacturer	
Ontology	In computer_science, an ontology is the attempt to formulate an exhaustive and rigorous conceptual schema within a given domain, a typically hierarchical data structure containing all the relevant entities and their relationships and rules (theorems, regulations) within that domain. The computer science usage of the term *ontology* is derived from the much older usage of the term in philosophy, where it means the study of being or existence as well as the basic categories thereof. See ontology (philosophy). Ontology languages. To be useful, ontologies must be expressed in a concrete notation. An ontology language is a formal language by which an ontology is built. There have been a number of data languages for ontologies, both proprietary and standards-based: The Cyc project had its own ontology language based on first-order logic, called CycL. KIF was, among other things, another ontology language. OWL is a language for making ontological statements, developed as a follow-on from RDF and RDFS, as well as earlier ontology language projects including OIL and DAML. *[wordIQ.com]*	
PDM	Product data management	
PLM	Product lifecycle management	
PPR (model)	product, process and resource (model)	

Pragmatics	Pragmatics is generally the study of natural language understanding, and specifically the study of how context influences the interpretation of meanings. It is a subfield of linguistics. The context here must be interpreted as *situation* as it may include any imaginable extralinguistic factor, including social, environmental, and psychological factors. *[wordIQ.com]*	
Productive system	A software application or system deployed for commercial purposes (here within the automotive industry) *[wordIQ.com]*	
Proposition	In modern logic, a **proposition** or **ansatz** is what is asserted as the result of uttering a sentence. In other words, it is the meaning of the sentence, rather than the sentence itself. Different sentences can express the same proposition, if they have the same meaning. *[wordIQ.com]*	
ProSAp	Process Step Application; here: step within product development	
QA	Quality assurance	
QAF	Quality Assurance Feature	
RAM	Random-access memory	
Representational element	Used within a representation formalism to represent informational entities of a certain kind.	information representation element, knowledge representation element
SA	Service application; a transparent extension of the ULEO server's functionality	
Scalability	The optimized adaptability to current needs and company-specific regulations.	
Scenario	A scenario, in the sense used in this thesis, is a collection of subsequent tasks to be performed in order to achieve certain milestones within the overall product development process. These tasks are performed by engineers on the basis of certain product data and using a given IT environment. Scenarios are not only used to discuss technical details, but also to show up the new way of work from the engineers', i.e., user's, point of view. The scenarios discussed in this thesis describe parts of the <u>new</u> and desired way of product development, rather than the current situation	
SCI modeling	Modeling of SCI (self-contained information). SCI is information that is equipped with meta-information on its semantics. Other term for SCI modeling is semantic modeling.	
Semantic network	A semantic network is often used as a form of knowledge representation. It is a directed graph consisting of *vertices* which represent concepts and *edges* which represent semantic relations between the concepts. Semantic networks are a common type of machine-readable dictionary. Important semantic relations are set out below: *Meronymy* (A is part of B)	

	Holonymy (B has A as a part of itself) *Hyponymy* (or *troponymy*) (A is subordinate of B; A is kind of B) *Hypernymy* (A is superordinate of B) *Synonymy* (A denotes the same as B) *Antonymy* (A denotes the opposite of B) *[wordIQ.com]*	
Semantics	In general, semantics (from the Greek *semantikos*, or "significant meaning," derived from *sema*, sign) is the study of meaning, in some sense of that term. Semantics is often opposed to syntax, in which case the former pertains to what something *means* while the latter pertains to the formal structure/patterns in which something is *expressed* (e.g., written or spoken). *[wordIQ.com]*	
Specific information		instance information
STEP	Standard for the Exchange of Product Model Data	
Syntax	The **first meaning** of the term **syntax** can be described as the study of the rules, or "patterned relations" that govern the way the words in a sentence come together. It concerns how different words (which, going back to *Dionusios Thrax*, are categorized as *nouns*, *adjectives*, *verbs*, etc.) are combined into *clauses*, which, in turn, are combined into *sentences*. In the earliest framework of semiotics (established by C.W. *Morris* in his **1938** book *Foundations of the Theory of Signs*) the **syntax** is defined within the study of signs as one of its three subfields, the first being **syntax** (the study of the interrelation of the signs), the second subfield being **semantics** (the study of the relation between the signs and the objects to which they apply), and the third subfield being **pragmatics** (the relationship between the sign system and the user). *[wordIQ.com]*	
Template (engineering ~)	A large and complex EO constellation, building block for product engineering, offered to the engineer by software	
UDF	User-defined feature	user feature
UFM	Unified feature model, predecessor of UMEO	
ULEO	Universal Linking of Engineering Objects; Note about the *terminology*: This research started off as an investigation into approaches for feature mapping, proceeded to feature linking, and has now reached the concept of universal linking of engineering objects. As it is not feasible to change names given for the approaches too often over time, the authors decided to keep the methods' names stated below, even if the notion of features has been replaced by that of engineering objects in the meantime. So, ULEO is largely still called feature linking. It is important to point out, though that every appearance of features is also valid for EOs.	
UMEO	Unified Model of Engineering Objects; part of the IIM	
User	A *user* is someone who utilizes a software application in order to fulfill a task as part of the product development process, e.g., concept design, detail design, manufacturing planning, assembly	

	planning, etc.	
User-defined feature	*User-defined features* (*UDFs*) are those defined and implemented by company-specific end-users or standardization committees, without any means of programming and rebuilding the CAx system. They usually represent intellectual property of a company.	UDF, user feature
User information	Information that is of relevance for its source and destination; other information is, for example, information wrapping the user information in packages for transport purposes.	
UXS	ULEO XML schema	
VDAFS	VDAFS (Vereinigung Deutsche Automobilindustrie Flächen Schnittstelle) is a German neutral file format for the exchange of surface geometry. It was developed to exchange free form surfaces and it became a DIN standard in 1986. VDAFS supports elementary curve and surface geometry entities and some topology to define more complex models. VDAFS is used in the German automotive industry to define surface models, e.g., car bodies. It was expected to be replaced by STEP by 1996. _[CERN*]_	
VMI	View meta-information	

Footnotes

[*] European Organization for Nuclear Research, see the URL _http://public.web.cern.ch/Public/Welcome.html_; for VDAFS see the URL _http://cadd.web.cern.ch/cadd/cad_geant_int/thesis/node33.html_

Chapter 23 Technical Background Information

This appendix collects basic information that is useful for the understanding of the contents of this thesis but does not belong to this research's contributions.

23.1 Object-oriented Feature Modeling

This section briefly examines the most common way of modeling features.

As indicated in numerous publications, feature technology has proven to provide major benefits for product development processes (see *[Haasis, 1997]*, *[Haasis et al., 2001]*, and *[Haasis et al., 2003]*).

The way features are modeled strongly affects the resulting benefits of feature technology. It has a direct impact on the quality of the information that can be represented inside product- and other PPR models and concretizes in terms of the collection of feature properties, the relations between features and the meta-information on both. Other quality **criteria** are extensibility of models and the option to integrate them into other models, as well as the absence of redundancy, the availability of building blocks for new features, and user-definability of features and relations. Otto Salomons gives a detailed overview of various a-priori and a-posteriori approaches to feature-based modeling and the consequences on feature representations (see *[Salomons, 1995]*).

Object-oriented Feature Modeling. The modeling of features in accordance with the object-oriented paradigm as proposed by Rumbaugh et al. (see *[Rumbaugh et al, 1991]*) seems to have become the standard method. This means that features are differentiated and arranged as feature types or classes. After this is done, feature classes can be instantiated into PPR models (→ feature instances). A feature class may consist of attributes, which describe the static properties, and sometimes also of methods, which describe the dynamic behavior of the respective feature instances. To reduce redundancy among feature classes, one or more steps of abstraction are typically performed. The result of finding feature classes, performing abstractions, and arranging the classes by linking them through specialization relations is a taxonomy of feature classes. Feature classes at the bottom level of the taxonomy tree can be instantiated by the user, whereas classes located higher in the hierarchy cannot.

23.2 Topic Maps

This section introduces topic maps and XTMs. They have been further discussed in the section presenting the state of the art.

XML topic maps[*] (XTMs) are models of information resources that are isolated from them but point to them (commonly using URIs; see *[XTMspec, 2001]* for the specification. See also Pepper and Garshol *[Pepper & Garshol, 2002]* for an example application). They focus more on the finding of information resources by providing meta-information on them and less on representing information (i.e., knowledge) on domains of the world (although topics maps could also be utilized for this purpose).

History. The original *topic maps paradigm*[†] was introduced in 1993 in the context of the Davenport Group (see *[Hunting & Park, 2002]*, *[Pepper, 2002]*). In an activity called *Conventions for the Application of HyTime*, the paradigm was further developed in the environment of the GCA

Footnotes

[*] See the URL *http://www.topicmaps.org/xtm/index.html*.

[†] See the URL *http://www.topicmaps.org/xtm/1.0*.

Research Institute (now known as IDE Alliance[*]). In 2000, the topic map paradigm was formalized as an ISO standard, ISO/IEC 13250:2000. Subsequently, the TopicMaps.Org Authoring Group (AG) was founded with the task to develop the applicability of topics maps to the World-Wide Web. The ISO standard XML Topic Map (XTM) was established in 2001, applying the XML standard for representing topic maps.

Various **commercial applications** of XTM are available:

~ Moresophy[†], for example, offers an "*integrated tool portfolio for design and operative utilization of knowledge nets*" called Toolset L4, which claims to use semantic nets and to be "*XTM compatible*".

~ *Empolis*[‡] offers the empolis knowledge management suite and claims to use the topic map standard for its solutions, applying "explicit ontologies and intelligent retrieval".

Topics. For this purpose, the XML Topic Maps standard provides a representation formalism including a formal semantic specification and a grammar. XTMs support the description of topics and the relationships between them, while topics can denote and represent a variety of entities such as objects or abstract concepts of the real world (called subjects) or relations between them (called associations) or types of subjects or relations (called topic types). Hence, topics represent a subject within a topic map (see Figure 48 for an example). Subjects may originate in the real-world domain but also within a topic map. As a consequence, multiple (meta-) levels of information can be represented within the same topic map.

Reification, Subjects, and Topic Characteristics. "*The act of creating a topic is called reification. When anything is reified it becomes the subject of the topic thus created*"[§].

Each topic may have the following characteristics: names (*topic names*), *roles* played within relationships (*associations*), and resources (*occurrences*). Such a set of a topic's characteristics is only valid within a dedicated *scope*, which could also be called a *context*.

Each topic can have zero to many types. Types, in turn, are also topics. The association of a type to a topic is also an association.

Footnotes

[*] IDE Alliance, International Digital Enterprise Alliance; see the URL *http://www.idealliance.org*.

[†] See the URL *http://www.moresophy.de*.

[‡] See the URL *http://www.empolis.de*.

[§] See the URL *http://www.topicmaps.org/xtm/1.0*.

```
<topicMap xmlns=http://www.topicmaps.org/xtm/1.0/
     xmlns:xlink="http://www.w3.org/1999/xlink" ID="jillstm">
     <topic ID="kudo">
            <!-- An instance of the "description occurrence" class -->
            <instanceOf>
                  <topicRef xlink:href="#description"/>
            </instanceOf>
            <baseName>
                  <baseNameString>Kudo</baseNameString>
            </baseName>
     </topic>
…
<!--........................ TOPIC TYPES ........................-->
     <topic ID="developer">
            <subjectIdentity>
                  <subjectIndicatorRef xlink:href="http://psi.ontopia.net/jill/#developer"/>
            </subjectIdentity>
            <baseName>
                  <baseNameString>Developer</baseNameString>
            </baseName>
     </topic>
     <topic ID="company">
            <subjectIdentity>
                  <subjectIndicatorRef xlink:href="http://psi.ontopia.net/xmltools/#Company"/>
            </subjectIdentity>
            <baseName>
                  <baseNameString>Company</baseNameString>
            </baseName>
     </topic>
</topicMap>
```

Figure 48: XTM Example: Topics, Topic Types, and Names [Ontopia[*]]

Scopes. A scope "*specifies the extent of the validity of a topic characteristic assignment*". Although a topic's set of characteristics is specific to a certain scope, the scope of the characteristics themselves is not regulated and left up to being interpreted by the software applications. Scopes can be specified explicitly as a set of topics, resources, or subject identifiers (see below). If they are specified implicitly (in fact, such a topic simply does not have any scope specification), they are called unconstrained scopes, which may be interpreted as globally valid.

Scopes are not specified for a single topic as a whole but individually for its single characteristics *baseName*, *occurrence*, and *association*.

Topic Names and Subject Identities. A ***topic*** can have more than one name within its context. One of them, the base name, has to be unique[†], while others (variant names) need not. Other names (such as display names or sort names) can be selected specifically for an individual context.

A ***subject***, reified into a topic, can be unambiguously identified by a subject indicator (called *subject descriptor*[‡] in the first specification). "*When two topics use the same resource to indicate their subject, they are by definition "about" the same thing, and must therefore be merged during processing*". Commonly, URIs are utilized as subject indicators. Identifiers contained in documents already published on the Internet can be referred to as *published subject indicators (PSIs)* or *published subject descriptors (PSDs)*.

Correlation of and mediating between several topic maps. Two or more topic maps can be conjoined to a single topic map after identifying topics referring to the same subject. This can be achieved by using the above-mentioned subject indicators. However, two topic maps can also be

Footnotes

[*] See the URL *www.ontopia.net/omnigator/docs/jill.xtm*.

[†] According to the Topic Naming Constraint

[‡] See the URL *http://www.topicmaps.org/xtm/1.0/#desc-subject-indicator*

correlated using topics with two subject indicators, each referring to a topic within one of the two topic maps to be joined.

Relations between topics. A topic association links one or more topics (*members*), each of which takes over a certain role *(association role)* within the relationship. These roles are represented within the association, as its name (*label*) is. A role may be specified by a role (*topic*) type, thus restricting possible role fillers. A topic may play different roles in different scopes (as a role is a topic characteristic, which, in turn, is scope dependent).

Occurrences are the resources that are to be described by the topic map. Each topic can refer to a resource as its occurrence. Each occurrence can be specified more closely by an occurrence role. As with association roles, occurrence roles can also be based on an (occurrence) role type.

23.3 The Semantic Web

This section introduces the Semantic Web, which has been elaborated in section Part III – 7.1.4 during the survey of the state of the art.

One of today’s largest information spaces is the *World-Wide Web* – and, as with the much smaller product development domain, one of the major problems is the fuzzy meaning and incompatible representation of information, mainly caused by a lack of an explicit representation of the WWW information’s semantics. One of the key reasons for this situation is, in turn, the lack of appropriate methods for representing such semantics. In order to tackle this problem, the World-Wide Web Consortium developed a package of means aimed at delivering semantic descriptions of web contents. This bundle consists of several components, and its development took several years. “*The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web.…On Feb 10, 2004, the World-Wide Web Consortium announced final approval of two key **Semantic Web** technologies, the revised **Resource Description Framework (RDF)** and the **Web Ontology Language (OWL)**.*” This statement can be found on the W3C’s home page[*] *[W3Ca, 2005]*. The W3C’s third main pillar is the **Extended Markup Language** (**XML)** (see *[W3C XML, 2000]* and *[W3C XMLschema, 2001]***).** The bundle is rounded off by the **XML metadata interchange format** (XMI). On the W3C web pages, the following concise characterization of the Semantic Web’s components is provided:

- ***“XML** provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.*
- ***XML Schema** is a language for restricting the structure of XML documents and also extends XML with datatypes.*
- ***RDF** is a datamodel for objects ("resources") and relations between them, provides a simple semantics for this datamodel, and these datamodels can be represented in an XML syntax.*
- ***RDF Schema** is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.*
- ***OWL** adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes”*.

Footnotes

[*] See the URL *http://www.w3.org/*.

23.3.1 Resource Description Framework, RDF

Focus. On the W3C web pages *[W3C RDFprimer, 2004]*, RDF is introduced as follows: *“The Resource Description Framework (RDF) is a language for representing information about resources in the World-Wide Web. … By generalizing the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web”. … RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers, or* ***URIs****), and describing resources in terms of simple properties and property values. … This enables RDF to represent simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values.”*

✍ The well-known Uniform Resource Locator (**URL**) is a character string that identifies a Web resource by representing its primary access mechanism (essentially, its network location). A URL is a special kind of URI.

The forms http: (Hypertext Transfer Protocol, for Web pages), mailto: (email addresses), ftp: (File Transfer Protocol), and urn: (Uniform Resource Names, intended to be persistent location-independent resource identifiers, as used, for example, in urn:isbn:0-520-02356-0 for a book) are examples of URI schemes (URI forms) that have already been developed for various purposes.

Generally, URIs are decentrally created by organizations and software vendors, although specific URI schemes are managed centrally such as http: for URLs (by DNS).

Furthermore, the W3C notes that *“RDF URIs can refer to any identifiable thing, including things that may not be directly retrievable on the Web.”* A URI, together with an optional fragment identifier at the end, which is separated by the "#" character, is called the URI reference (**URIref**). RDF employs URIrefs to identify individuals or things, kinds of things (concepts), characteristics, and values of properties. W3C defines a resource as *“anything that is identifiable by a URI reference, so using URIrefs allows RDF to describe practically anything, and to state relationships between such things as well”*. There are two RDF notations: RDF graphs (see Figure 50) and the machine processable RDF/XML (see Figure 49). Both reflect RDF’s underlying notion that things (subjects) are described by named properties (predicates), which have values (objects).

```
<?xml version="1.0"?>
<rdf:RDF    xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
            xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

     <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
            <contact:fullName>Eric Miller</contact:fullName>
            <contact:mailbox rdf:resource="mailto:em@w3.org"/>
            <contact:personalTitle>Dr.</contact:personalTitle>
     </contact:Person>

</rdf:RDF>
```

Figure 49: Example for Application of an RDF Type [W3C RDFprimer, 2004]

RDF schemas (RDF-S) allow specification of vocabularies, i.e., type systems, for RDF (see Figure 51). Such type systems are similar to those of object-oriented languages (instances of classes, taxonomies of classes, inheritance), but RDF-Ss are considered to be additional information about the RDF resources described and do not restrict them. For example, attributes of instances do not have to possess values.

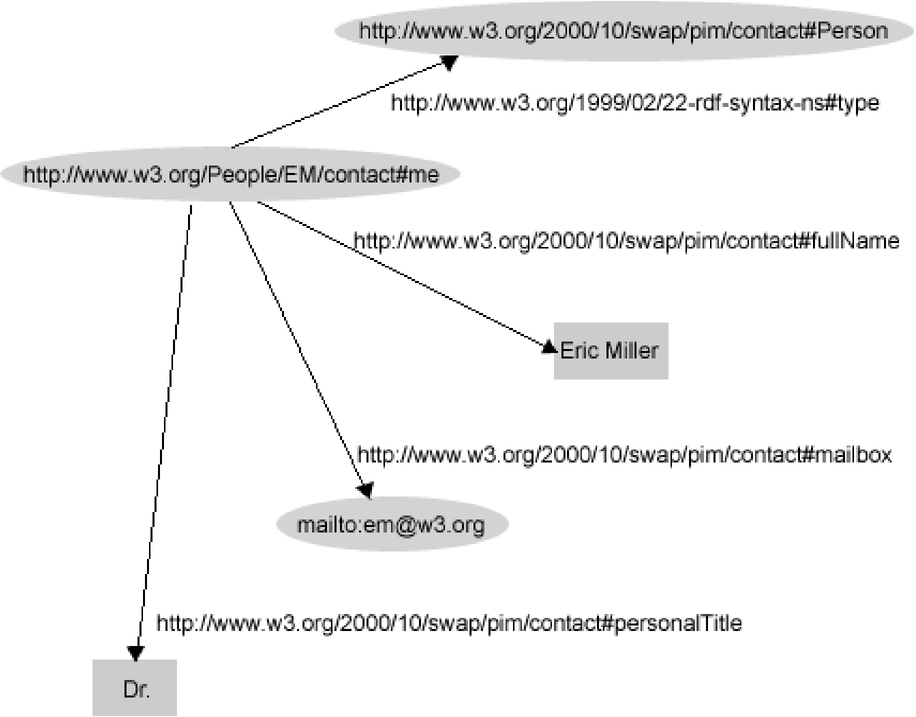

http://www.w3.org/2000/10/swap/pim/contact#Person
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/People/EM/contact#me
http://www.w3.org/2000/10/swap/pim/contact#fullName
Eric Miller
http://www.w3.org/2000/10/swap/pim/contact#mailbox
mailto:em@w3.org
http://www.w3.org/2000/10/swap/pim/contact#personalTitle
Dr.

Figure 50: Example RDF Graph [W3C RDFprimer, 2004]

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xml:base="http://example.org/schemas/vehicles">

     <rdf:Description rdf:ID="MotorVehicle">
            <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
     </rdf:Description>

     <rdf:Description rdf:ID="PassengerVehicle">
            <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
            <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
     </rdf:Description>

     <rdf:Description rdf:ID="Truck">
            <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
            <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
     </rdf:Description>

     <rdf:Description rdf:ID="Van">
            <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
            <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
     </rdf:Description>

     <rdf:Description rdf:ID="MiniVan">
            <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
            <rdfs:subClassOf rdf:resource="#Van"/>
            <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
     </rdf:Description>

</rdf:RDF>
```

Figure 51: An RDF-S Class Hierarchy [W3C RDFprimer, 2004]

23.3.2 Web Ontology Language, OWL

OWL (see *[W3C OWL, 2004]*) has been developed by revising the web ontology language *DAML+OIL* (see *[W3C DAML+OIL, 2001]*). It provides an additional vocabulary on top of RDF and RDF-S and possesses a formal semantics, with the intention to describe the meaning of concepts and their instances. There are three sublanguages of OWL, which are equipped with an increasing expressive power: *OWL Lite*, *OWL DL*, and *OWL Full*. W3C explains: *"The first level above RDF required for the Semantic Web is an ontology language that can formally describe the meaning of terminology used in Web documents. … OWL has been designed to meet this need for a Web Ontology Language. …* ***OWL Lite*** *supports those users primarily needing a classification hierarchy and simple constraints. …* ***OWL DL*** *supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). …* ***OWL Full*** *is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees."*

OWL concepts (classes, see Figure 52) are described through **properties** (see Figure 53). OWL provides three types of relations called properties: data-valued properties (slots in the frame system's terminology), individual-valued properties (concept-correlating), and annotation properties. The type and the number of entities referred to by properties can be restricted (see Figure 54). Properties can be arranged hierarchically (see Figure 53).

```
<owl:Class rdf:ID="Wine">
     <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
     <rdfs:label xml:lang="en">wine</rdfs:label>
     <rdfs:label xml:lang="fr">vin</rdfs:label>
</owl:Class>

<owl:Class rdf:ID="Pasta">
     <rdfs:subClassOf rdf:resource="#EdibleThing" />
</owl:Class>
```

Figure 52: OWL Class [W3C OWLb, 2004]

```
<owl:Class rdf:ID="WineDescriptor" />

<owl:Class rdf:ID="WineColor">
     <rdfs:subClassOf rdf:resource="#WineDescriptor" />
</owl:Class>

<owl:ObjectProperty rdf:ID="hasWineDescriptor">
     <rdfs:domain rdf:resource="#Wine" />
     <rdfs:range rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
     <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
     <rdfs:range rdf:resource="#WineColor" />
</owl:ObjectProperty>
```

Figure 53: OWL Property Hierarchy [W3C OWLb, 2004]

```
<owl:Class rdf:ID="Wine">
     <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
     <rdfs:subClassOf>
            <owl:Restriction>
                  <owl:onProperty rdf:resource="#madeFromGrape"/>
                  <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
            </owl:Restriction>
     </rdfs:subClassOf>
</owl:Class>
```

Figure 54: OWL: Cardinality of Properties [W3C OWLb, 2004]

```
<owl:Thing rdf:ID="CentralCoastRegion" />

<owl:Thing rdf:about="#CentralCoastRegion">
     <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

Figure 55: Incremental Definition of an OWL Individual [W3C OWLb, 2004]

OWL enables concepts and individuals to be defined incrementally (Figure 55), i.e., using several statements. Additionally, set-oriented class definitions combined with logical operators are possible (axioms).

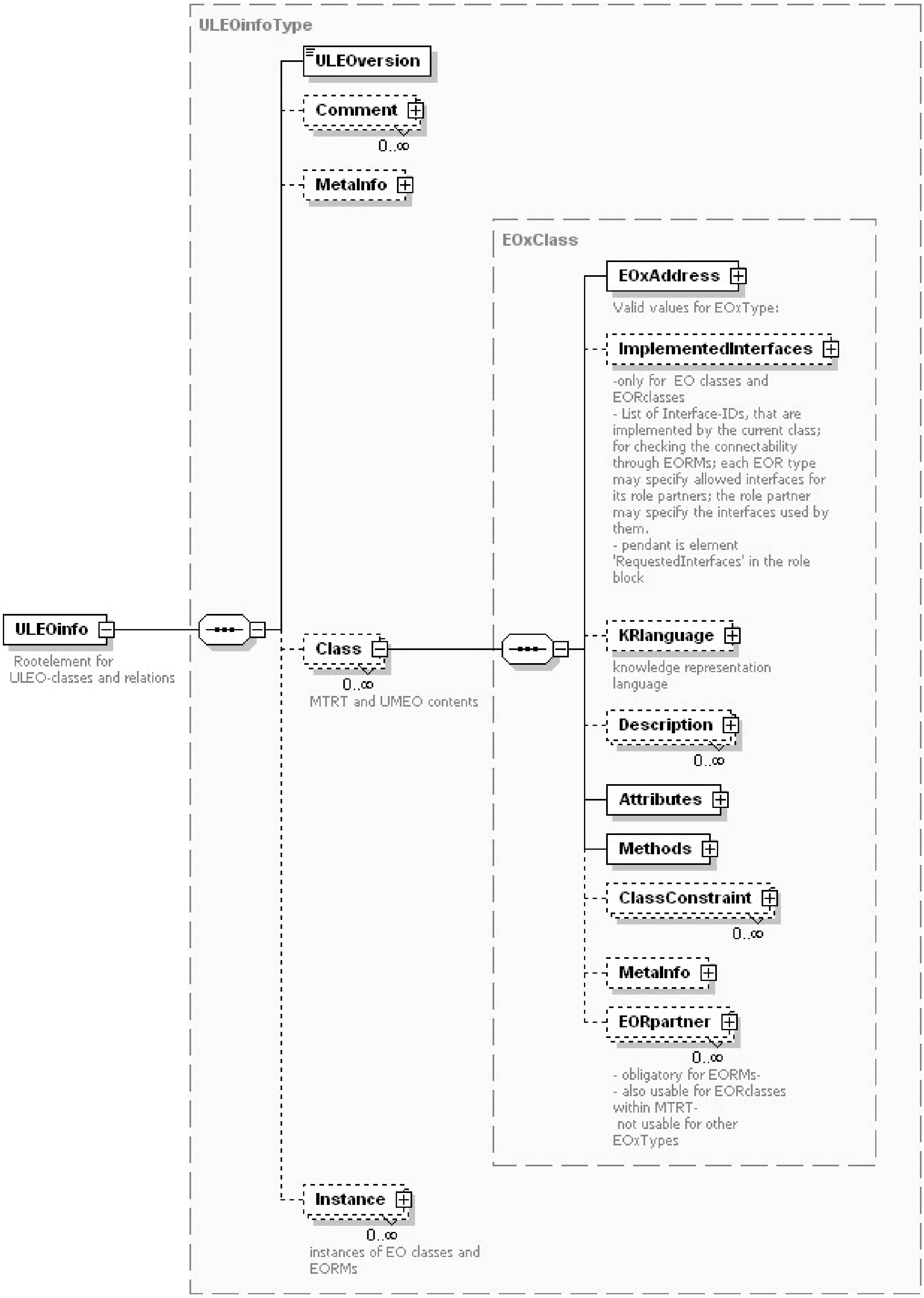
Ontology Mapping. Concepts and their instances, stored within multiple ontologies can be mapped onto each other by declaring them "equivalent" or "same". Also, instances carrying identical names can be declared different from each other. Multiple **versions** of ontologies can be distinguished (see Figure 56).

```
<owl:Ontology rdf:about="">
     <owl:priorVersion rdf:resource="http://www.w3.org/TR/2003/CR-owl-guide-
20030818/wine"/>
</owl:Ontology>
```

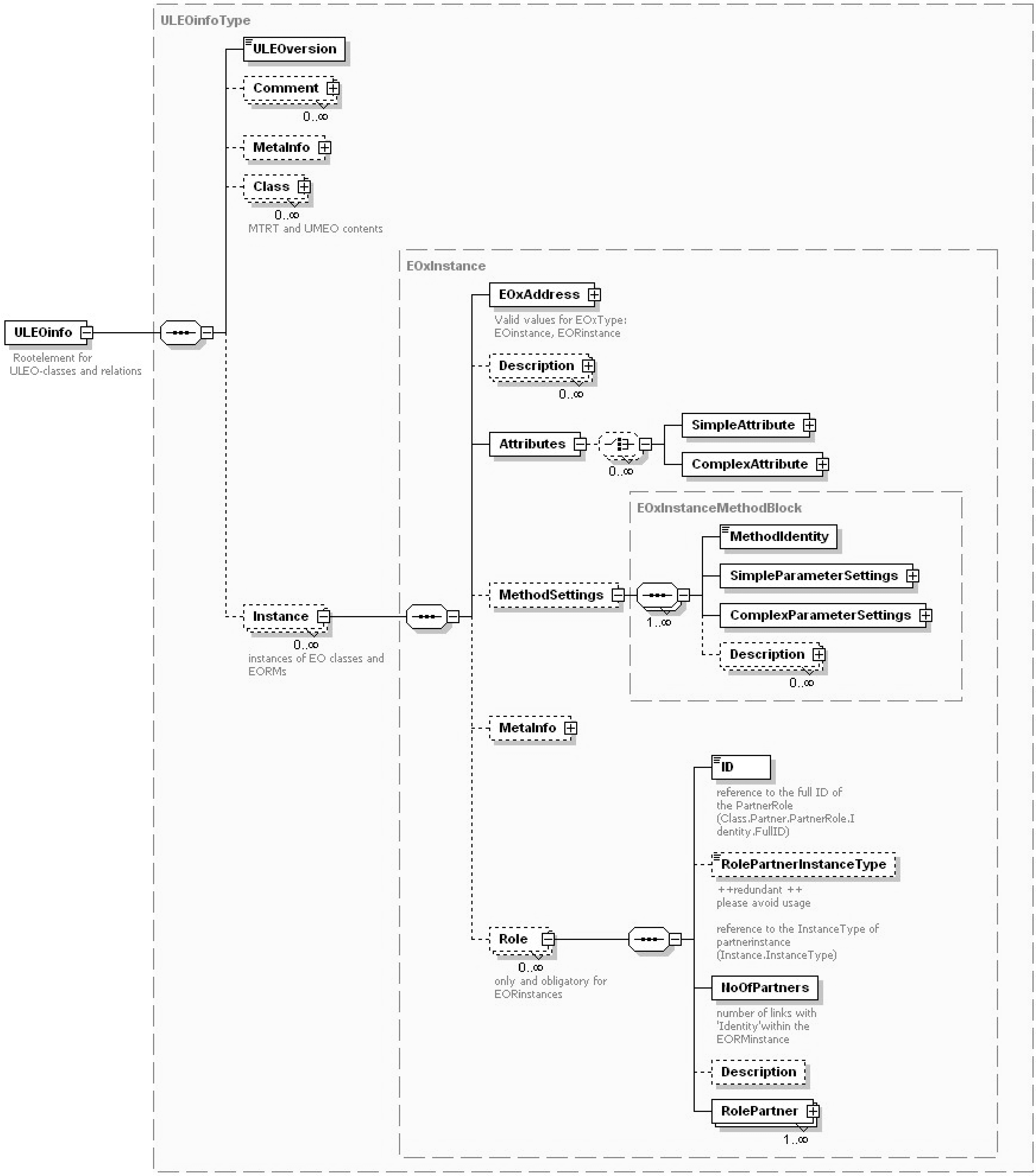
Figure 56: Ontology Versioning in OWL [W3C OWLb, 2004]

Chapter 24 ULEO XML Schema

In addition to the illustrations in the text above, this appendix illustrates some major extracts from the ULEO XML schema.


ULEOinfoType
ULEOversion
Comment
0..∞
MetaInfo
EOxClass
EOxAddress
Valid values for EOxType:
ImplementedInterfaces
-only for EO classes and EORclasses
- List of Interface-IDs, that are implemented by the current class; for checking the connectability through EORMs; each EOR type may specify allowed interfaces for its role partners; the role partner may specify the interfaces used by them.
- pendant is element 'RequestedInterfaces' in the role block
ULEOinfo
Rootelement for ULEO-classes and relations
Class
0..∞
MTRT and UMEO contents
KRlanguage
knowledge representation language
Description
0..∞
Attributes
Methods
ClassConstraint
0..∞
MetaInfo
EORpartner
0..∞
- obligatory for EORMs-
- also usable for EORclasses within MTRT-
not usable for other EOxTypes
Instance
0..∞
instances of EO classes and EORMs

ULEO XML – Class Element for Abstract Information



ULEO XML – Instance Element for Specific Information

Chapter 25 Services of the ULEO IPCI

This appendix lists some details of the most relevant ULEO GIS services (ULEO Inter-Process Communication Interface, ULEO IPCI).

25.1 MEO Services

These are services for accessing and editing a Model of Engineering Objects, *which is equivalent to IIM/UMEO. The central-entrance MEO is the* Unified Model of Engineering Objects (UMEO), *which is administrated by the ULEO server.*

Service Signature	Description
`ULEOreturnCode` **`Add_XML`**`(SessionId aSessionId, ULEOXML aULEOXML, boolean aOverWrite)`	Adds EOclasses and EORMs contained in aULEOXML; returns a code identifying the result of the operation.
`EOxAddress` **`GetRootClassAddress`**`(SessionId aSessionId, AddressContext aContext)`	* Returns EOxAddress of the highest ranking EOclass in UMEO.
`EOxAddressArray` **`GetDirectSuperClassAddresses`**`(SessionId aSessionId, EOxAddress aEOclass)`	Returns Array of EOxAddresses with all direct super-EOclasses.
`EOxAddressArray` **`GetDirectSubClassAddresses`**`(SessionId aSessionId, EOxAddress aEOclass)`	Returns Array of EOxAddresses with all direct sub-EOclasses.
`ULEOinfo` **`GetEOclassFullContents_XML`**`(SessionId aSessionId, EOxAddress aEOclass, boolean LocalClassOnly)`	Returns the XML of an EOclass; if local, then the result will only contain local attributes and methods; if not local, then the result will also include inherited attributes and methods.
`IdArray` **`GetEOclassAttributeNames`**`(SessionId aSessionId, EOxAddress aEOclass, boolean LocalClassOnly)`	Returns attribute names of the EOclass identified by aEOclass; if local, then the result will only contain local attributes names; if not local, then the result will also include inherited attributes names.
`IdArray` **`GetEOclassMethodNames`**`(SessionId aSessionId, EOxAddress aEOclass, boolean LocalClassOnly)`	* Returns method names of EOclass; if local, then the result will only contain local attributes names; if not local, then the result will also include inherited attributes names.
`EOxAddressArray` **`GetEORMsForEOclass`**`(SessionId aSessionId, EOxAddress aEOclass, EOxAddressArray aMTRTclassIdsFilter, EOxId aRoleNameFilter, boolean LocalClassOnly)`	Returns array of EOxAddresses of the relations of which the EOclass identified by aEOclassId is a partner; if local, then the result will only contain local relations; if not local, then the result will also include inherited relations. If a list of MTRTclassIds is given as a filter, the result will contain EOxAddresses of EORMs that are of a type included in the given list. aRoleNameFilter specifies that the given class aEOclassId has to be connected to the desired EORMs by a fixed role name. Wildcards (*) may be included inside EOxAddress.
`EOxAddressArray` **`GetEORMsForEORM`**`(SessionId aSessionId, EOxAddress aEORMid, EOxAddressArray aMTRTclassIdsFilter, EOxId aRoleNameFilter, boolean LocalClassOnly)`	Returns array of EOxAddresses of the relations of which EORM identified by aEORMid is a partner; if local, then the result will only contain local relations; if not local, then the result will also include inherited relations. When a list of MTRTclassIds is given as a filter, the result will contain EOxAddresses of EORMs that are of a type included in the given list. aRoleNameFilter specifies that the given EORM aEORMid has to be connected to the desired EORMs by a fixed role name.
`ULEOinfo` **`GetEORMsForEOx_XML`**`(SessionId aSessionId, EOxAddress aEOxAddress, EOxAddressArray aMTRTclassAddressFilter,`	Returns all relations of the EO class or EORM identified by aEOxAddress as XML; if local, then the result will only contain local attributes and methods as

EOxId aRoleNameFilter, boolean LocalClassOnly)*	well as local relations; if not local, then the result will also include inherited attributes, methods, and relations. When a list of MTRTClassIds is given as a filter, the result will contain the EORMIds of the EORMs that are of a type included in the given list.
ULEOinfo **GetSubTreeForEOclass_XML**(SessionId aSessionId, EOxAddress aEOclassId)	Returns EOclass and all relations of the EOclass identified by aEOclassId as well as all EOclasses and relations below in XML.
EOxAddress **CreateEOclass_XML**(SessionId aSessionId, ULEOXML aEOclassXML, EOxAddress aSuperEOxAddress)	Creates a new EOclass; returns the EOxAddress of the EOclass created.
ULEOreturnCode RemoveEOclass(SessionId aSessionId, EOxAddress aEOclassId)	Removes EOclass identified by aEOclassId; returns a code identifying the result of the operation.
ULEOreturnCode **UpdateEOclassAddress**(SessionId aSessionId, EOxAddress aEOclassAddress)	Updates EOclassAddress (EOxAddress is not updated); returns a code identifying the result of the operation.
ULEOreturnCode **UpdateEOclass_XML**(SessionId aSessionId, EOxAddress aEOclassId, ULEOXML aEOclassContent)	* Updates EOClass; if aEOClassId is not equal to the Id in ULEOXML, then an error is returned; returns a code identifying the result of the operation.
ULEOinfo **GetEOclassContext_XML**(SessionId aSessionId, EOxAddress aEOclassId, boolean DirectSuperClassesOnly, EOxAddressArray aEOSubClassIdsFilter, EOxAddressArray aMTRTclassIdsFilter)	Returns the EOclass identified by aEOclassId and all its super- and subclasses and relations that are set in the corresponding filter parameters as one UMEO XML block.
ULEOreturnCode **AddSuperClassesOfEOclass**(SessionId aSessionId, EOxAddress aEOclassId, EOxAddressArray aSuperClassOfEOclassIdArray)	Adds super classes to the EOclass identified by aEOclassId; returns a code identifying the result of the operation.
ULEOreturnCode **RemoveSuperClassesOfEOclass**(SessionId aSessionId, EOxAddress aEOclassId, EOxAddressArray aSuperClassOfEOclassIdArray)	* Removes super-classes of the EOclass identified by aEOclassId; returns a code identifying the result of the operation.
ULEOreturnCode **ReplaceSuperClassOfEOclass**(SessionId aSessionId, EOxAddress aEOclassId, EOxAddress aCurSuperClassOfEOclassId, EOxAddress aNewSuperClassOfEOclassId)	* Replaces super-class of the EOclass identified by aEOclassId; returns a code identifying the result of the operation.
ULEOinfo **GetEORMfullContents_XML**(SessionId aSessionId, EOxAddress aEORMid, boolean LocalClassOnly)*	Returns EORM in XML; if local, then the result will only contain local attributes and methods; if not local, then the result will also include inherited attributes and methods.
EOxAddressArray **GetEORMpartnerAddresses**(SessionId aSessionId, EOxAddress aEORM, EOxId aRoleNameFilter)	Returns EOxAddress of all EOxes that are partners of the EORM identified by aEORM.
EOxAddressArray **GetEOpartnerAddressesForEORM**(SessionId aSessionId, EOxAddress aEORM, EOxId aRoleNameFilter)	Returns array of EOClassAddresses of all partner EOClasses of the EORM identified by aEORMid.
EOxAddressArray **GetEORMpartnerAddressesForEORM**(SessionId aSessionId, EOxAddress aEORM, EOxId aRoleNameFilter)	Returns array of EORMAddresses of all partner EORMs of the EORM identified by aEORMid.

ULEOreturnCode **RemoveEORM**(SessionId aSessionId, EOxAddress aEORM)	Removes EORM identified by aEORMid; returns a code identifying the result of the operation.
EOxAddress **MaterializeEOR**(SessionId aSessionId, ULEOXML aULEOXML)	Creates a new EORM; returns Id of EORM created.
ULEOreturnCode **UpdateEORMAddress**(SessionId aSessionId, EOxAddress aEORMAddress)	* Updates EORMAddress (EOxAddress is not updated); returns a code identifying the result of the operation.
ULEOreturnCode **UpdateEORM_XML**(SessionId aSessionId, EOxAddress aEORM, ULEOXML aEORMcontent)	Updates EORM; if aEORM does not match the address given inside ULEOXML, then an error will be returned; returns a code identifying the result of the operation.
EOxClassAttributeDeclaration **SetAttributeDeclaration**(SessionId aSessionId, EOxAddress aClassAdd, Identifier Attributename, XMLstring value)•	
EOxClassAttributeDeclaration **GetAttributeDeclaration** (SessionId aSessionId, EOxAddress aClassAdd, String Attributename, locator = “defaultparam.german.short”)	Returns the part of the class declaration specifying the requested attribute.
AttributeOrMethodContent **GetContentsViaEOxPath**(SessionId aSessionId, (eoxstartid), EOxPath oPath)	To access contents of attributes and methods of EOx via path notation, e.g., “(this).role1.attribute2”, e.g., to change UMEO-external references.
ULEOresult **CallEOxMethodViaEOxPath**(eoxstartid, EOxPath)	For example, to evaluate UMEO-external references.

25.2 MTRT Services

These are services for accessing and editing the Meta-taxonomy of Relation types (MTRT).*The MTRT hosts the types of all EO relations within IIM/UMEO.*

Service Signature	**Description**
EOxAddress **GetRootClassAddress**(SessionId aSessionId, UserId aUserId)	returns MTRTclassAddress of the highest ranking MTRTclass in MTRT
EOxAddressArray **GetDirectSuperClassAddresses**(SessionId aSessionId, EOxAddress aMTRTclassId)	Returns MTRTclassIds of all direct super-MTRTclasses of the MTRTclass identified by aMTRTclassId
EOxAddressArray **GetDirectSubClassAddresses**(SessionId aSessionId, EOxAddress aMTRTclassId)	Returns MTRTClassIds of all direct sub-MTRTclasses of the MTRTclass identified by aMTRTclassId
ULEOinfo **GetMTRTclass_XML**(SessionId aSessionId, EOxAddress aMTRTclassId, boolean LocalClassOnly)	Returns MTRTclass in XML; if local, then the result will only contain local attributes and methods; if not local, then the result will also include inherited attributes and methods
ULEOinfo **GetMTRTclassContext_XML**(SessionId aSessionId, EOxAddress aMTRTclassId, boolean DirectSuperClassesOnly, EOxAddressArray aMTRTclassIdsFilter)	Returns MTRTclass identified by aMTRTclassId as one UMEO XML block and, if DirectSuperClassesOnly, then those are included according to the given filter.
ULEOinfo **GetSubTreeForMTRTclass_XML**(SessionId aSessionId, EOxAddress aMTRTclassId)	Returns MTRTclass identified by aMTRTClassId and all inherited MTRTclasses in XML
ULEOreturnCode **Add_XML**(SessionId aSessionId, ULEOXML aULEOXML)	Adds MTRTclasses and MTRTrelations (currently only single inheritance) contained in aULEOXML; returns a code identifying the result of the operation
ULEOreturnCode **RemoveMTRTclass**(SessionId aSessionId,	Removes MTRTClass identified by aMTRTClassId; returns a code identifying the result of the operation

EOxAddress aMTRTclassId)	
EOxAddress **CreateMTRTclass**_XML(SessionId aSessionId, ULEOinfo aULEOXML, EOxAddress aSuperMTRTclassId)	Creates a new MTRTClass; returns Id of MTRTClass created
ULEOreturnCode **UpdateMTRTclassAddress**(SessionId aSessionId, EOxAddress aMTRTclassAddress)	Updates MTRTclassAddress (aEOxAddress is not updated); returns a code identifying the result of the operation.
ULEOreturnCode **UpdateMTRTclass_XML**(SessionId aSessionId, EOxAddress aMTRTclassId, ULEOXML aMTRTclassContent)	Updates MTRTClass; if aMTRTClassId is not equal to the Id in ULEOXML, then an error is returned; returns a code identifying the result of the operation
IdArray **GetMTRTclassAttributeNames**(SessionId aSessionId, EOxAddress aMTRTclassId, boolean LocalClassOnly)	Returns attribute names of the EOclass identified by aEOxAddress; if local, then the result will only contain local attributes names; if not local, then the result will also include inherited attributes names.
IdArray **GetMTRTclassMethodNames**(SessionId aSessionId, EOxAddress aMTRTclassId, boolean LocalClassOnly)	Returns method names of the MTRTclass identified by aMTRTclassId; if local, then the result will only contain local attributes names; if not local, then the result will also include inherited attributes names
ULEOreturnCode **ReplaceSuperClassOfMTRT**(SessionId aSessionId, EOxAddress aMTRTclassId, EOxAddress aCurSuperClassOfMTRTid, EOxAddressNewSuperClassOfMTRTid aNewSuperClassOfMTRTid)	Replaces super class of MTRTclass identified by aMTRTclassId; returns a code identifying the result of the operation

25.3 EOx Instance Service

These are services for accessing and editing instances of EO classes and EORMs.

Service Signature	Description
ULEOinfo **GetEOinstFullContents_XML**(SessionId aSessionId, EOxAddress aEOinstAddress)	Returns XML of one or more Eo instances (wildcard: '*')
EOxAddressArray **GetEOinstAddresses**(SessionId aSessionId, EOxAddress aEOinstAddress)	Returns addresses of all EO instances specified by aEOinstAddress; may contain wildcards, "*".
ULEOreturnCode **WriteEOxInst_XML**(SessionId aSessionId, ULEOXML aInsertEOxInst_XML, EOxAddress aDestAddress)	Creates or updates one or more instances of EO classes or EORMs
ULEOreturnCode **RemoveEOxInst**(SessionId aSessionId, EOxAddress aEOinstAddress)	Deletes one or more instances of EO classes or EORMs. aEOinstAddress may contain wildcards, "*"
EOxAddressArray **GetEORIsForEOxInstance**(SessionId aSessionId, EOxAddress aEOxInstAddress, EOxAddressArray aMTRTclassFilter, boolean LocalClassOnly)	Returns array of EOxAddresses of the relation instances of which EOxInstance identified by aEOxInstAddress is a partner (aEOxInstAddress may designate an EO instance or an EORinstance). A MTRTclassFilter may restrict the types of relations considered
EOxAddressArray **GetEORIpartnerAddress**(SessionId aSessionId, EOxAddress aEORIaddress, ULEOstringList aPathnames)	Returns addresses of all EOxInstances (EOR instances or EO instances) that are partners of the EOR instances (EORI) identified by aEORIaddress
ULEOinfo **GetEOinstAttributesAndValues**(SessionId aSessionId, EOxAddress aEOinstAddress)	Returns XML of attribute names and values from an already instantiated EO.
ULEOreturnCode **SetEOinstAttributesAndValues**(SessionId aSessionId, EOxAddress aEOinstAddress)	Sets attibute values of an EO instance; returns a code identifying the result of the operation

Part VIII – INDEXES

Chapter 26 Index of Figures

Chapter 27 Table Index

Chapter 28 Keyword Index

Part IX – To Finish

This research work shaped several years of my life. And although I gained a considerable number of new insights, I must admit that I still agree with what Socrates said as cited in the beginning of this thesis: "*I know that I don't know anything – and even that I hardly know*". Nevertheless, when looking at ULEO, I feel satisfaction, and I am confident that its ideas will contribute to improving product development.

Diese Forschungsarbeit prägte mehrere Jahre meines Lebens. Und obwohl ich eine Fülle neuer Erkenntnisse gewonnen habe, muß ich gestehen, daß ich immer noch mit dem eingangs zitierten Wort Sokrates' übereinstimme: „Ich weiß, daß ich nichts weiß – und selbst das weiß ich kaum". Dennoch verbinde ich mit ULEO ein gewisses Maß an Zufriedenheit, und ich bin zuversichtlich, daß dieser Ansatz dazu beitragen wird, die Produktentwicklung voranzubringen.

Steinheim, in January 2005

Johann U. Zimmermann

Research carried out within the Laboratory of Design, Production, and Management embraces the manufacturing of industrial products and focuses on developments in computer-aided manufacturing, covering the overall range from design to the integral control of the activities on the shop floor. Over the last decade, the integrated approach towards the product realization process has become a necessity as – for various reasons – industry has increasingly been experiencing problems with the implementation of part-solutions. Reports of the research projects are distributed in a limited edition by the Laboratory of Design, Production and Management. The series is published with the ISSN number 1386-5307. Dissertations that have been released previously are listed below:

A.H. van ‘t Erve	Generative computer aided process planning for part manufacturing, an integrated approach	1988
J.R. Boerma	The design of fixtures for prismatic parts	1990
F.J.A.M. van Houten	PART: a computer aided process planning system	1991
J.J. Tiemersma	Shop floor control in small batch part manufacturing	1991
F.J.C.M. Jonkers	A software architecture for CAPP systems	1992
H.J.W. Vliegen	Classification systems manufacturing; managerial control of process knowledge	1993
A. Lenderik	The integration of process and production planning in small batch part manufacturing	1994
L.J. de Vin	Computer aided process planning for the bending of sheet metal components	1994
R.M. Boogert	management in computer aided process planning Tool	1994
O.W. Salomons	Computer support in the design of mechanical products	1995
A.L. Arentsen	A generic architecture for factory activity control	1995
R. Geelink	Flexible definition of form features	1996
J. de Vries	Integrated process planning for small batch manufacturing of sheet metal components	1996
J.H. Kappert	Integration of component design, process planning and die design in rubber pad forming	1997
A. Liebers	An architecture for cost control in manufacturing	1998
R.E. Begelinger	Computer support in the design of product families	1998
P.A. Wollf	Conceptual Design of Warships	2000
M.M.T. Giebels	EtoPlan; A concept for concurrent manufacturing planning and control	2000
D. Lutters	Manufacturing integration based on information management	2001
T.H.J. Vaneker	Development of an integrated design tool for aluminum extrusion dies	2001
S. Finke	Solid freeform fabrication of metal components by extrusion and deposition of semi-solid metals	2002
E. Ten Brinke	Costing support and cost control in manufacturing	2002
D. Wijnker	Integration of information in manufacturing systems	2003
A. Layer	Case-based Cost Estimation – A Building Block for Product Cost Management and Design-for-X	2004
R. Mentink	Process Management in Design & Engineering; Applying dynamic process modeling based on evolving information content	2004
B. M. Sailer	Market-oriented Order Planning in the Automotive Industry. A Building Block for Support of Efficient Order Processing	2004