

ULEO – Universal Linking of Engineering Objects

J. U. Zimmermann^{1,2}, S. Haasis¹, F.J.A.M. van Houten² (1)

¹DaimlerChrysler Research and Technology, Dept. of Product, Process, Resource Integration RIC/EP, Ulm, Germany

²University of Twente, Dept. of Engineering, Laboratory for Design, Production and Management, Enschede, The Netherlands

Abstract

This paper reports on ongoing research in the field of feature-based product development. The resulting ULEO approach aims at enabling a high-quality flow of information between applications and at universal automation of product model generation. This is achieved by modeling all classes of relevant objects within a Unified Model of Engineering Objects (UMEO). All types of relationships are modeled inside a dedicated meta taxonomy of relation types and materialize inside the UMEO. Informational relations represent ontological knowledge about dependencies between object classes and are suited for cross-linking of product models. Generative relations describe knowledge on how to create instances automatically.

Keywords:

Product development, information flow, automation

1 INTRODUCTION

This contribution describes ongoing research in the field of development process chain integration in the automotive industry. After a brief introduction to the research area and stating and motivating the goals in the first two chapters, previous approaches are discussed in chapter 3. In chapter 4 the new ULEO approach is formulated. To motivate the solutions chosen, existing approaches are contrasted to the authors' goals. Then the main thesis is formulated and the key concepts of ULEO are discussed. Chapter 5 summarizes the major issues and points out further investigations.

Each application along the product development process chain (**process step application - ProSAp**) has to fulfill specific tasks using specialized technical knowledge and modi operandi for problem solving. As a result, each ProSAp has its own view of the product. IT solutions must adapt to these different characteristics in order to be able to represent effective and intuitive tools and aids for the users. 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. While domain-specific feature types have a variety of benefits, there is also a down-side: the use of specialized feature types may lead to a logical separation of the feature-based models produced. Scientific approaches covering **feature mapping** functionality – also called **feature conversion** or **feature transformation** systems – try to tackle this problem in various manners, 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 features of A and B.

2 GOALS AND MOTIVATION

Pure feature mapping, doesn't 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. This research aims at finding a solution for feature linking, thereby enabling a high-quality flow of information between applications and a universal automation of product model generation.

3 PREVIOUS WORK

First, some comments will be given on several scientific approaches which are relevant to the field of feature linking.

Characteristic: Direct or Indirect Mapping

There are systems, such as Suh's [1] and Wong's [2], which convert feature-based models indirectly via an intermediate model that contains geometry-oriented¹, application-neutral intermediate feature classes and instances. Deviating from such neutral-model approaches, Bronsvort [3] and De Kraker [4] use the design feature model as the central intermediate model. 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 set of 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 which is relevant along the process chain. As a consequence, intermediate features may be considered all-purpose feature classes, which is cumbersome and leads to well-

¹ The term *geometry-oriented* as used here means that every intermediate feature focuses on representing product geometry, although it may also contain other information.

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.

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 about this, 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**² to arrange the feature classes. For instance, while Shah [5], Krause [6-7], and Srikantappa [8] refrain from using taxonomies at all, De Kraker [4] and Lecluse [9] do use them. 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 authors. Wong [2] uses separate taxonomies of feature classes for the individual feature-based applications. 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 objects inside the same taxonomy.

Characteristic: How 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 which 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 applications. Another 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 [6-7], for example, sticks to this method, using a representation language called PDGL³ to express the knowledge inside the feature classes. The most straightforward solution, method (b1), is in part used, for example, by Shah [10] where user-defined design features are mapped on generic machining features via a hard-coded CFT⁴ reconstruction algorithm. 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 [10] also uses approach (b2) for pre-defined design features. The system interprets

² Hierarchy of feature classes based on inheritance relations.

³ Part Design Graph Language

⁴ Constructive Feature Tree

production rules to map them onto machining process operations by choosing alternatives from a pre-defined machining process graph. Han [11] uses rules to weight mapping hints inside design features. Wong [2] converts neutral features into application-specific features using production rules. 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 [12], for example, uses “codes” inside a design-feature-based model to derive downstream models. So far, the authors' literature studies have not yielded any true examples for (b3)-like systems. Conceptual graphs however, such as those used by Geelink [13] for other purposes (feature definition and recognition) are an option for handling information about objects and their inter-relations in a clear fashion.

Characteristic: Generation and Maintenance of Links Between Feature Sets

A further characteristic of feature mapping systems is the generation and maintenance of links between the source and destination feature sets. In other words, 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 automation aspects. Although it would be theoretically possible to provide for this, e.g. in the approach taken by Krause [6-7], there is no evidence that this has been done thus far. Suh's papers [1] and [14], which are further examples of intermediate-model approaches, claim 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 object at all. In Bronsvort's paper [3] 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.

Other Characteristics

Although there are other characteristics which are fundamental for the motivation of the ULEO approach, they will just be touched on here, largely because they get less attention in the literature. For example, the following aspects have been skipped: **knowledge processing**, modeling of **feature constellations**, **complexity** and **cardinality** of possible relations between the mapped feature sets, **classified relations**, support of **user-defined features**, aspects of **concurrent engineering**, **mapping on demand vs. mapping on the fly**.

To conclude, 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.

4 THE ULEO APPROACH

The next paragraphs will introduce the approach called Universal Linking of Engineering Objects (ULEO).

Engineering objects (EOs) are informally introduced here as any objects relevant in engineering, such as assemblies, features, parts, surfaces, tolerances, materials, etc.

4.1 Conclusions Drawn from Existing Approaches

This section constitutes the link between existing approaches and ULEO.

Characteristic: Direct or indirect 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 **which does not use intermediate models** is proposed.

Characteristic: How Features Are Modeled

The capability to model all classes of features and other relevant objects inside a single global model is a prerequisite to being able to relate these entities to each other in a clear and straightforward fashion and to reason about them within the same context. A bit less formally put one could say that these entities know about each other.

Characteristic: How Mapping Knowledge Is Modeled

The combination of methods for storage of knowledge about mapping within special relations (which interconnect 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 knowledge 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).

Characteristic: Generation and Maintenance of Links between Feature Sets

Links between feature sets are the carriers of information for inter-ProSAP communication. In this respect, existing feature-based systems are also not able to offer a solution.

4.2 Main Thesis

The goals specified above can be reached by modeling engineering objects and their corresponding interrelations as follows (see Figure 1):

- (1) All EO classes are modeled within a central taxonomy called **Unified Model of Engineering Objects (UMEO)**. “Central” means that there is only one taxonomy for all ProSAPs, and the taxonomy is stored in only one globally accessible location.
- (2) All **types** of relationships between EO classes and/or instances are modeled inside a dedicated **meta taxonomy of relation types (MTRT)**, which is situated logically one abstraction layer above UMEO.

Relation types can be inheritance, aggregation or **engineering object relations (EORs)**, which are specialized directed associations. MTRT covers **informational EORs (IEORs)** as well as **generative EORs (GEORs)**.

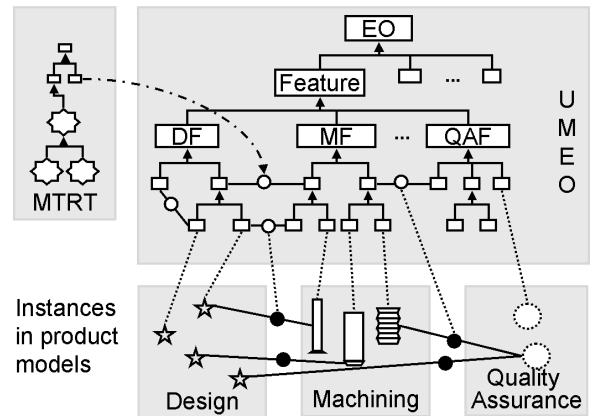


Figure 1: UMEO classes and instances.

4.3 Unified Model of Engineering Objects and Engineering Object Relations

UMEO has been introduced in order to allow all kinds of application-specific, standardized, and non-standardized EOs to be arranged in a unique and globally accessible taxonomy. This ensures that all the applications accessing the model get the same information and use the same concepts. Through the common model they all use the same notions. In addition, 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 like parts, assemblies, and resources. This increases the value of the model 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. Information on relations between objects or object classes is modeled in terms of dedicated relation objects, denoted as round black or white symbols in Figure 1. This permits relation-centered information to be kept outside the related objects or object classes. Thus, relations between the latter may change, while the related EOs and EO classes remain unchanged. Also, each EOR is of a specific type. In order to structure the possibly large quantity of EOR types, a special MTRT is introduced: its elements materialize inside UMEO as relations on the EO class level and instantiate inside the individual product models as relations on the EO level. As a result, cross-linked product models arise (see lower third of Figure 1). As EOR materializations are an integral part of UMEO, they are also inherited from parent to child EO classes, so that models remain as clear as possible. MTRT allows new relation types to be easily derived from existing ones using inheritance; and existing service applications such as cost estimation can roughly estimate the semantics of derived relation types by knowing their parent classes. The two main types of EORs cited above, *informational EORs* and *generative EORs* will be introduced. Both enable structured and targeted modeling of knowledge, so that it is directly related to the respective classes and objects inside the model. This matches the feature philosophy of attaching information exactly to what it refers to, thereby avoiding unintelligible all-in-one and hard-to-maintain knowledge bases. Moreover, using

inheritance together with EO classes makes it possible to apply EORs which reference some parent EO class also to its child classes. This is an elegant way of deducing new, explicit knowledge from the implicit. General knowledge is attached to parent classes and exceptions to their children.

Informational EORs (IEORs) describe logical relationships between EO classes, e.g. a relation between n design features and m machining features, which might be called "is_machined_as" and could describe which machining features are necessary to machine the related design features. Moreover, with IEORs, UMEMO may be augmented gradually by ontological knowledge, thus adding sophisticated information about the EOs' semantics, usage, and embedding within extended contexts of other applications' knowledge, for example. EO classes and IEOR classes can be instantiated into application-specific product models. IEOR instances interrelate individual product models by linking the logically corresponding EO instances inside of them, permitting information to be interchanged between ProSAPs on the feature level. Hence, IEORs have the potential to facilitate informational⁵ integration of the applications along the process chain. They may be used by application programs for a wide variety of purposes, such as design-to-X (for example cost estimation, manufacturability checks), feedback from downstream to upstream applications, etc.

Generative EORs (GEORs) describe knowledge on how to instantiate EOs and EORs automatically, for example triggered by and taking into account conditions inside product models. Examples of the latter are prior instantiations of features and their parameter values. Interpretation of these links results in automation of engineering tasks: thus, GEORs represent generative engineering knowledge. They provide a functionality subsuming feature mapping but are, in contrast, not limited to **feature** classes – in fact they can consider any engineering object. After each modification of a specific product model, the linking algorithm can be triggered to locate the class of the manipulated object inside UMEMO and to search for GEORs related to it. The GEORs' contents are interpreted by the system. These could be, for example, values of feature parameters or the number of new instances to be created. Since UMEMO's contents are meant for many different applications, they only consist of information about **what** is to be done, not **how** this is to be done. So, linking (mapping) knowledge may be changed without having to change ProSAP-specific elements.

5 SUMMARY AND FURTHER WORK

This paper outlines the key ideas of an approach called ULEO. ULEO generalizes the concept of features to that of engineering objects and focuses on the modeling and usage of typed relations between them. Thus, problems in the fields of not only automation but also process integration can be tackled. Information is stored locally and directly related to objects or object classes. The result is a clearly structured and maintainable knowledge base. ULEO is an open approach and sophistication of knowledge representation inside its models is scalable, which permits both backward-compatibility to older feature-based systems and also integration of and cooperation with company-wide information systems. The approach provides the base technology for further service applications. Process step applications are enabled to broaden their informational context, providing engineers with the information necessary for well-founded decision-

making. Thus, ULEO supports cooperation implicitly and explicitly, offering a rich variety of information stemming from various applications along the process chain.

Currently, several prototype implementations are under further development. They are evaluated through discussions with potential users within different application fields. For example, the following scenarios are being investigated: detail design of cylinder heads and mold tools is targeted together with the corresponding machining planning. Within this context, a special case of ULEO has been developed: EO constellations are sets of EO classes, related to each other through EOR classes. Users may instantiate all members of an EO constellation within a single action – this also includes instantiations into several product models such as finish-part and machining model. The EO instances' parameter values are synchronized. Another scenario considered for implementation is inspection planning for the automotive body in white

6 REFERENCES

- [1] Suh, Y. S., Wozny, M. J., 1998, Interactive Feature Extraction for a Form Feature Mapping System, Rensselaer Polytechnic Institute Troy, New York.
- [2] Wong, T. N., Leung, C. B., 2000, An object-oriented neutral feature model for feature mapping, *Internat. Journal of Production Research*, 38:3573-3601.
- [3] Bronsvort, W. F., Noort, A., van den Berg, J., Hoek, G. F. M., 2001, Product development with multiple-view feature modelling, *Proc. FEATS 2001*.
- [4] De Kraker, K. J., 1997, Feature Mapping for Concurrent Engineering, Delft University of Technology, Delft (NL), Doctoral Thesis.
- [5] Shah, J. J., 1988, Feature transformations between application-specific feature spaces, *Computer-Aided Engineering Journal*, 5/6:247-255.
- [6] Krause, F.-L., Kramer, S., Rieger, E., 1991, PDGL. A Language for Efficient Feature-Based Product Gestaltung, *Annals of the CIRP*, 40/1:135-138.
- [7] Krause, F.-L., Ulbrich, A., Vosgerau, F. H., 1991, Feature Based Approach for the Integration of Design and Process Planning Systems, J. P. A. M. W. J. Turner (Ed.), *IFIP*, 285-297.
- [8] Srikantappa, A. B., Crawford, R. H., 1992, Intermediate Geometric and interfeature relationships for automatic group technology part coding, *ASME* 1992, 245-251.
- [9] Lecluse, 1999, A design support system for three dimensional components and assemblies, *Lehven, Belgium, Doctoral Thesis*.
- [10] Shah, J. J., Hsiao, D., Leonard, J., 1993, A Systematic Approach for Design-Manufacturing Feature Mapping. In M. J. W. A. M. J. P. P.R. Willson (Ed.), *Geometric Modeling for Product Realization. IFIP 1993*, 205-221.
- [11] Han, J. H., Requicha, A. A. G., 1997, Modeler-independent feature recognition in a distributed environment. *Computer Aided Design report*, 30/6:453-463.
- [12] Ishii, K., Miller, R. A., 1992, Design representation for manufacturability evaluation in CAD. *Beyond feature-based design, ASME* 1992, 37-44.
- [13] Geelink, R., 1996, Flexible definition of form features. University of Twente, Enschede (NL), Doctoral Thesis.
- [14] Suh, Y. S., 1995, A Feature-Conversion CAD System for the Concurrent Engineering Environment, Rensselaer Polytechnic Institute, Troy, New York, Doctoral Thesis.

⁵ In the sense of on the information level.