Projection-Based Knowledge Representation for Concurrent Engineering

Carsten Thomas *
Daimler-Benz AG
Alt-Moabit 96a
10559 Berlin, Germany
thomas@dbresearch-berlin.de

Jerzy W. Rozenblit
Dept. of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721, U.S.A.
jr@ece.arizona.edu

ABSTRACT

This paper describes an integrative approach to represent knowledge structures and entities in the concurrent engineering enterprise. General types of knowledge are classified and presented in various perspectives and abstractions. A new concept of knowledge representation based on *projections* is introduced. It is demonstrated how the projection concept integrates various facets of CE. Conclusions discuss the benefits of the proposed representation scheme and discuss potential application scenarios.

1 INTRODUCTION

Concurrent Engineering (CE) [1] paradigm has emerged in response to increased global competition in product design, deployment, and marketing. The main characteristic of CE is the integration of all development activities and as well as participating teams through the entire product life cycle. In a holistic perspective, CE is both an organizational and technological effort. In this paper, we focus on the technological aspect, namely, on the adequate knowledge representation to support the concurrent engineering process.

In the existing CE systems, general purpose knowledge representation schemes such as semantic nets are used [2]. However, these representations do not provide efficient methods for knowledge acquisition and handling.

In this paper, we introduce a new formal means of knowledge representation that overcomes the limitations of the current schemes by combining the strengths of both, general purpose and specialized representations. The new scheme is intended to (i) address the variety of knowledge aspects, (ii) help structure CE knowledge in a clear and concise manner, (iii) support maintaining knowledge consistency and avoiding knowledge redundancy.

To achieve these goals, we propose to use an object-centered approach in combination with a new method for the differentiated access to the acquired knowledge. The new representation scheme is called *Projection-Based Knowledge Representation.*

2 CONCURRENT ENGINEERING KNOWLEDGE

This section characterizes knowledge facets that are predominant in CE.

Structuring Engineering Knowledge

In Concurrent Engineering processes, an overwhelming amount and variety of knowledge is available and has to be used. To be represented, the knowledge used in the engineering processes has to be structured through an ordering scheme and a means to capture the interdependencies among the different characteristics of knowledge.

We propose a fourfold structuring of the engineering knowledge:

- A piece of knowledge belongs to one of three *general types of knowledge*: knowledge about an engineering domain, knowledge about the specific product to be designed and produced, and knowledge about how to design and manufacture products.
- A piece of knowledge serves a certain *perspective* of the design (e.g., it contains geometric, temporal, or behavioral knowledge).
- A piece of knowledge describes something on a certain *level of abstraction*.
- A piece of knowledge has a certain *maturity*.

In the following, we will review these characteristics in more detail.

*This research has been conducted while the first author was on a sabbatical at The University of Arizona, Department of Electrical and Computer Engineering.

General Types of Knowledge: In engineering processes, three general types of knowledge are typically employed:

Domain knowledge is available in the domain from previous engineering processes or as common knowledge. It consists of standards information, knowledge about common characteristics of products of a domain, results from earlier engineering work or knowledge abstracted thereof.

Product knowledge (also referred to as "instance knowledge") contains information on properties of the engineering process output. Typical examples for product knowledge are design data, construction plans, CAD files, implemented control software and similar information.

Process knowledge holds information about the properties of the process that leads to the desired engineering output. It comprises information about how and when to process product knowledge, about engineering step dependencies or their prerequisites and alike, and additional information such as timing and resource constraints that stem from external requirements.

Perspectives: The knowledge available about products and processes can be structured in a manner that focuses on an single, individual area of interest, the so-called *perspective*. Typical perspectives used in CE are "system behavior", "geometry", "finance", "system safety", and alike.

To express facts and rules, and to formulate operations to be used within a perspective, *languages* may be used. These languages usually are different for each perspective, but also have common subsets to support the handling of knowledge interdependencies among different perspectives.

If a piece of knowledge describes a complex part of a product or process, them it may reference knowledge pieces that describe its components. The actual decomposition of products or processes may be different for individual perspectives. Within a perspective, aspects can be evaluated by operations. The information may be generated from (i) a single fact available under the perspective, (ii) an interpretation of a more general fact, (iii) knowledge about the decomposition of the product or process into components and the interrelations between its components.

Levels of Abstraction: All knowledge about products and processes is available on various levels of abstraction. The descriptions may be available as abstract specifications, conceptual design sketches, detailed construction plans, and alike. Moreover, there may be several different refinements for a less detailed specification, each focusing on a specific detail of the system. Although often not accounted for in traditional engineering environments, there are strong relations between abstract and detailed knowledge pieces describing the same product or process. It should be possible to define formal mappings between abstract and detailed versions, and thus to test for the validity of refinements or abstractions.

Maturity: The knowledge acquired or produced in the engineering process changes over time. The design of products and processes will change reflecting the stepwise approach to ever better engineering results and the change of engineering goals. However, knowledge from earlier design steps, which is not up-to-date anymore, should also be available in later engineering steps. It contains experience and design solutions which might be useful later on.

Existing Knowledge Representation Schemes

Due to the heterogeneity and the amount of knowledge that is dealt with in CE, there are many very complex relations and interdependencies between the individual pieces of knowledge. These relations do exist not only within knowledge types, perspectives, and abstraction levels, but they also involve all knowledge characteristics. Therefore, existing knowledge representations that focus only on certain general types knowledge (like the Domain Modeling Method [3] and schemes based on the System Entity Structure concept [4] for domain knowledge), or on certain perspectives (like various exchange formats for geometric design data) lack the representational adequacy for CE.

3 KNOWLEDGE REPRESENTATION BASED ON PROJECTIONS

In this section we propose a new scheme that integrates a variety of knowledge facets and thus provides a richer representation than the ones used in current CE environments.

Knowledge Entities and Projections

We term our approach *object-centered* since we focus on the knowledge that is available for individual objects of the real world. The knowledge is aggregated within the so-called *knowledge entities*. Formally, knowledge entities are set theoretic structures whose elements contain parameters, constraints, rules, formal specifications, and references to other knowledge entities. We structure the knowledge entities along the three dimensions: a) domain knowledge, b) instance knowledge, and c) process knowledge.

Individual knowledge entities are identified by their names. Relations between knowledge entities are expressed by using the entity identifiers as references. The knowledge entities concentrate the knowledge available for the individual objects. To deal with such knowledge in detail, the entities have to be viewed under certain *projections*. Then, they reveal their inner structure, their decompositions, and their parameterization.

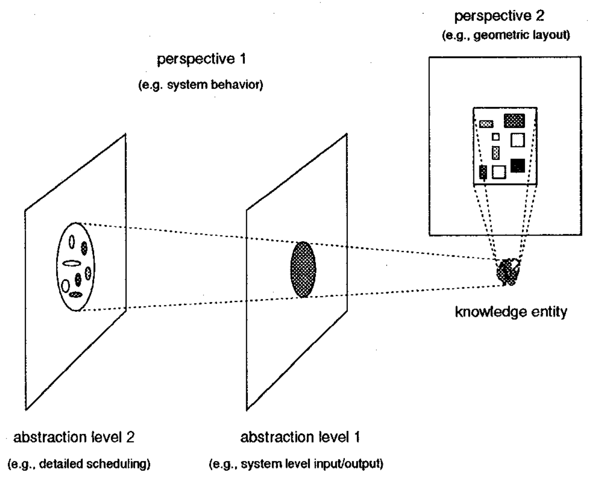

Figure 1: Knowledge projections under different perspectives at various levels of abstraction

Projections are combinations of perspectives and levels of abstraction (figure 1). The visible details of a knowledge entity depend on the chosen projection. Specifications or parameters that are visible under one projection might not be accessible under another one. Typically, also decompositions and component interdependencies are different for some of the projections.

Projections are used to acquire and handle knowledge in a given context. In a projection operation, the knowledge that is available about an object for a given perspective and abstraction level is extracted from the knowledge entity and is transformed into an arbitrary conventional representation that is commonly used in this context. For projections, all knowledge is held directly within the knowledge entity (i.e., no decomposition exists for this projection).

If decompositions exist in a certain projection, then the complete knowledge about the object in this projection is computed as a combination of the respective projections of all components. In order to retrieve the necessary knowledge, the knowledge entities of the components are evaluated under the same projection as the decomposed object. Then, the result of this evaluation can be combined according to rules that are specified as part of the knowledge entity of the decomposed object.

The concept of *knowledge entities and projections* differs from other approaches in two main aspects:

- *All knowledge about an object is concentrated in a knowledge entity.* Thus, a concise knowledge representation and the avoidance of redundancy is achieved.
- *Projections provide differentiated access to the knowledge* (taking into account the desired perspective and level of abstraction) and hide all information that is not needed in the current context.

CE Environment Properties

The formal representation of knowledge about products and processes should be viewed in a specific context that provides the semantics for the representation. The perspectives and levels of abstraction in a specific environment depend on the tools that are available, on the nature of the products to be engineered, and on the engineering processes. Therefore, the properties of CE environments are captured to provide a formally defined context for the application of the knowledge representation scheme.

Formally, the capabilities of a Concurrent Engineering environment (CEE) are denoted by the following structure

$$CEE = \langle \mathcal{A}, \mathcal{P}, \{\lambda^p\}, \{\Omega^p\} \rangle,$$

where $\mathcal{A}$ is the set of pre-declared abstraction levels, $\mathcal{P}$ is the set of pre-declared perspectives, λ^p is a language to formulate expressions to be used in a particular perspective $p \in \mathcal{P}$, and Ω^p is a set of applicable operations, formulated using the language of this perspective.

Domain Knowledge Entities — DKE

Domain knowledge describes variants of objects under certain perspectives and in certain levels of abstraction. In the projection-based approach, all knowledge available about an object and its variants is contained in the *Domain Knowledge Entity*. Formally, a domain knowledge entity DKE is a structure

$$DKE = \langle b, \Pi, \Delta, \{c_\pi\}, \{D_\pi\}, \{I_\pi\}, \{M_\pi\} \rangle.$$

A domain knowledge entity is either an independent specification, or specializes another DKE. If it is a specialization, this is denoted by the identifier b of the more general DKE. In this case, the structure elements extend or specialize the elements of the base entity. An independent DKE is indicated by the replacement of b by the "empty" symbol $(\cdot)$.

A DKE is defined for the projections listed in the set of permissible projections $\Pi \subseteq A \times P$, with $A \subseteq \mathcal{A}$ and $P \subseteq \mathcal{P}$. For these projections, the operations $\omega \in \Omega^p$ (as defined for perspective p with $(a,p) = \pi \in \Pi$ in the environment capabilities) can be carried out. The knowledge necessary for these operations is either contained in this DKE or can be extracted from DKE referenced as components in decompositions of this DKE.

Usually, more than one decomposition is applicable to an object. While some of the differences are due to the different focus of the individual projections, other differences specify individual object decomposition variants to be used within one projection. To represent this fact, all decompositions applicable in a projection π are referenced in the decomposition set D_π of this projection. However, for some operations, the decompositions of different perspectives and/or abstraction levels have to be used in conjunction (e.g., to verify the appropriateness of a refinement, or to map functional components into a geometric layout). Here it is necessary to have a "global" notion of the decompositions that might be used. Therefore, a global set Δ of permissible decompositions is held independent from the projections.

Decompositions are lists of components. How these components actually interact depends on the projection that is applied to the knowledge entity. To be able to use decompositions in more than one projection, the additional information that describes component interrelations is held in so-called models. There is exactly one set of models M_π for each projection. This set of models contains one model for each decomposition that is applicable in the current projection: $\forall d \in D_\pi : \exists m_{\pi,d} \in M_\pi$ If the set of decompositions for this projection is empty, then this object is a terminal (atomic) object under this projection and the model contains all knowledge necessary in this projection: $D_\pi = \emptyset : M_\pi = \{m_\pi\}$.

Both, decompositions and models, can be parameterized in order to express multiple decompositions and their couplings. Also, this would allow to parameterize properties specified by models in terminal projections.

For each projection, an interface specification (cut) c_π does exist that provides a uniform access method to the knowledge contained in the decomposition and the model, no matter what decomposition is actually used.

In the engineering process, the domain knowledge contained in DKE is transferred into instance knowledge by selecting single decompositions for each projection and by parameterizing all variables of decompositions and models. The selection and generation rules for this process are held implicitly as declarative knowledge in form of the so-called invariants $i \in I_\pi$ that exist for each individual projection. Instance knowledge entities that are generated from DKE have to obey these invariants.

For the specification of both, models $m_{\pi,d}$ and invariants $i \in I_\pi$, the language λ^p of the perspective p is used, with $(a,p) = \pi \in \Pi$.

Instance Knowledge Entities — IKE

An instance knowledge entity (IKE) describes a subsystem of the engineered product. It contains all the knowledge about that subsystem. An IKE is denoted by a structure

$$IKE = \langle b, \{\overline{d_\pi}\}, \{\overline{m_\pi}\}\rangle.$$

In this structure, the base domain knowledge entity b is referenced to denote that this IKE is an instance of the domain knowledge contained in this individual DKE. For every projection $\pi \in \Pi^b$ of the domain knowledge entity b, the structure contains exactly one parameterized decomposition $\overline{d_\pi}$ (or the "empty" symbol for terminal projections) and one parameterized model $\overline{m_\pi}$. Of course, the original versions d_π and $m_{\pi,d}$ of $\overline{d_\pi}$ and $\overline{m_\pi}$ have to be members of the sets of applicable decompositions and models of the projection π, respectively.

For the instance knowledge entity, the following rules have to be observed: (i) For every projection, a decomposition and an appropriate model have to be selected and parameterized such that all invariants of the projection are satisfied (this includes also the invariants inherited from the predecessor DKE of b). (ii) If a certain decomposition is chosen in more than one projection, the invariants of all involved projections must hold. (iii) Variables in decompositions, models, and invariants which have the same identifiers are bound to the same value.

Process Knowledge Entities — PKE

Process knowledge entities are used to describe which engineering step has to be invoked to generate or extract knowledge from other knowledge. However, engineering steps are fixed sequences of sub-steps or basic operations on the available knowledge. In PKE, the prerequisites and the results of engineering steps are specified. Depending on the degree of generality of the invocation conditions, this can take two different forms:

First, PKE can specify generic dependencies between models that are available for certain projections. Here, nonempty sets of projection identifiers define what kind of knowledge is needed to start an engineering step and what knowledge will be available after its completion.

Second, PKE may be used to specify dependencies between certain models in DKE. In such PKE, nonempty sets of model identifiers specify which models have to be available before the engineering step is invoked, and which models will be generated.

Both forms, generic as well as specific PKE, define the engineering step in terms of a sequence of engineering sub-steps, or in terms of operations. In the latter case, the engineering step is described as a single operation or a fixed sequence of operations. The difference between sequences of engineering sub-steps and sequences of operations is that the elements of sub-step sequences have a general meaning in itself and can be individually applied wherever their invocation conditions are fulfilled. The operations have no general meaning (in terms of engineering steps) by itself, and must be grouped in predefined sequences.

From these two differentiations, we receive four forms of PKE:

$$\begin{aligned} PKE &= \langle {}^{\cdot}\Pi, \Pi^{\cdot}, \mathrm{seq}\,\omega, \ldots \rangle \\ PKE &= \langle {}^{\cdot}\Pi, \Pi^{\cdot}, \mathrm{seq}\,pke, \ldots \rangle \\ PKE &= \langle \{{}^{\cdot}m\}, \{m^{\cdot}\}, \mathrm{seq}\,\omega, \ldots \rangle \\ PKE &= \langle \{{}^{\cdot}m\}, \{m^{\cdot}\}, \mathrm{seq}\,pke, \ldots \rangle \end{aligned}$$

Here, ${}^{\cdot}\Pi$ and $\Pi^{\cdot}$ are the sets of projections that are needed as prerequisite and that are available as a result of an engineering step. Accordingly, ${}^{\cdot}m$ and $m^{\cdot}$ denote the respective sets of DKE models. The PKE engineering step itself is described either as a sequence of operations (seq ω) or as a sequence of PKE (seq *pke*). The latter construct allows us to specify engineering steps as aggregations of engineering substeps and thus supports the handling of information for process segments on different levels.

It might be useful to include invariant specifications into the PKE structure that are used to specify in detail the conditions that have to be fulfilled prior to the invocation of an engineering step. Other useful information to be included in a PKE are the facts about resource consumption, etc. information. The "..." symbolize that such extensions might be introduced later on.

Process Data Entities — PDE

Process data entities (PDE) are instantiations of the generic, domain-oriented process knowledge that is contained in PKE. They describe an engineering step that has to be undertaken to transform instances of knowledge (described by models that are part of IKE) into other instances. This is done to iteratively generate a network of instance knowledge entities that completely describe a product on the necessary level of detail. Therefore, PDE contain sets of references to the parameterized IKE models that are the prerequisites or results of the engineering step. Both, temporal and resource consumption information can be formulated as data or as constraints. The latter is necessary to formulate restrictions that apply only to the current project and may not be associated with PKE. Examples for such restrictions are project deadlines and project cost considerations.

Formally, process data entities are a structure

$$PDE = \langle {}^{\cdot}pde, pde^{\cdot}, \{{}^{\cdot}\overline{m}\}, \{\overline{m}^{\cdot}\}, \ldots \rangle$$

where ${}^{\cdot}pde$ and $pde^{\cdot}$ are the predecessor and successor PDE of this process data entity, $\{{}^{\cdot}\overline{m}\}$ and $\{\overline{m}^{\cdot}\}$ are the sets of model instances that have to be available prior or that are available as result of the engineering step, and "..." once again symbolize possible extensions of the structure.

Like process knowledge entities, process data entities may be grouped to specify larger engineering steps and project phases. For this purpose, an alternative process data entity structure has been defined:

$$PDE = \langle {}^{\cdot}pde, pde^{\cdot}, \{{}^{\cdot}\overline{m}\}, \{\overline{m}^{\cdot}\}, \mathrm{seq}\,pde, \ldots \rangle,$$

where seq *pde* denotes the sequence of PDE this process step consists of.

4 IMPLEMENTATION SCENARIO

In the past, much work has been done to provide computational support for concurrent engineering [5]. This includes the development of standardized data exchange and knowledge interchange formats, and tool invocation mechanisms. This work provides the technical basis for the interaction between engineering tools. However, to actually co-operate in achieving an engineering solution, the tools must have a common notion about the contents of the knowledge they operate on and their interrelations. This common basis for the tool interaction is provided by the framework of knowledge entities.

To take advantage of the distributed character of the knowledge representation scheme, we propose an accordingly distributed implementation of the CE environment. Such an environment consists of a number of interacting software programs, so-called (software) agents. Some of these agents (the *Supervisor Agents*) control knowledge entities, handle the knowledge contained in them and ensure their integrity. Other agents (the *Mediator Agents*) realize the communication between supervisor agents and traditional software tools (operating on their own local knowledge) that are used in the individual projections. The agents form a network of interacting software programs that communicate with each other in order to exchange knowledge and to effect operations (figure 2).

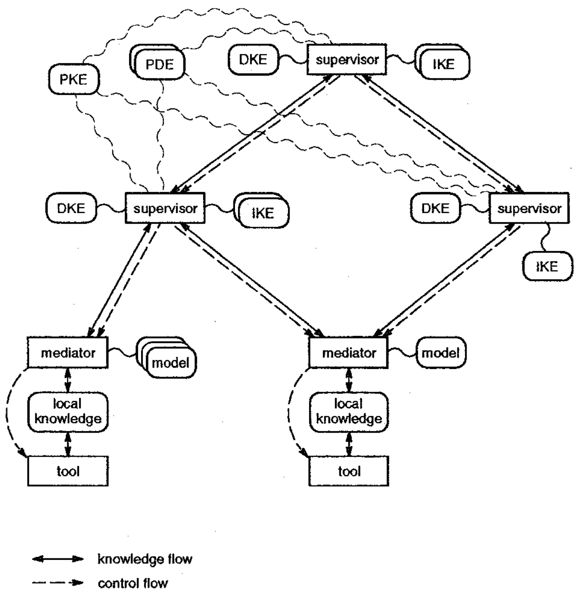

Figure 2: Partial structure of a projection-based CE environment

Contrary to the blackboard concept applied in most CE environments, this approach uses the structure inherent to the network of knowledge entities as a framework to organize the co-operation between the software agents and for the distribution of the engineering knowledge. This potentially allows to define which agents have access to certain parts of the knowledge and to partition and distribute the "commonly used knowledge base".

5 SUMMARY

We have proposed a set theoretic, integrative approach to represent knowledge requisite in the concurrent engineering enterprise. As opposed to the traditional techniques which use a variety of representations and descriptions to capture various facets of the CE-based product life cycle, our scheme affords a comprehensive description of the multiplicity of knowledge views and perspectives. As it is well grounded in a formal specification, knowledge integrity validation will be possible throughout the product life-cycle. In addition, this formal, underlying foundation is the basis for an agent-based, software realization of CE environment. We envision the following application scenarios where the projection-based representation would be particularly beneficial:

In the area of *system fault prevention and diagnosis*, the possibility to combine different views onto the system can be efficiently utilized for the generation of failure models, fault symptom tables, and test patterns during the design of a product, as well as for model-based diagnosis of existing systems. Here, models specifying the operational and faulty behavior of the system and safety constraints are envisioned to work together. For *management information systems*, the continuously and automatically updated process flow information together with the ability to retrieve product related information on the required level of detail provides a new quality of information access.

For our future work, we will consider the development of knowledge integrity validation procedures and further specification of the projection-based CE environment.

REFERENCES

[1] A. Kusiak, *Concurrent Engineering: Automation, Tools, and Techniques.* New York, NY, USA: John Wiley and Sons, 1993.

[2] D. Sriram and R. D. Logcher, "The MIT Dice project," *Computer*, vol. 26, pp. 64–65, Jan. 1993.

[3] H. Gomaa, "An object-oriented domain analysis and modeling method for software reuse," in *Proceedings of the International Conference on Systems Science, Hawaii*, vol. 2, (Los Alamitos, CA, USA), pp. 46–56, IEEE CS Press, Jan. 1992.

[4] J. W. Rozenblit and B. P. Zeigler, "Design and modeling concepts," in *International Encyclopedia of Robotics, Applications and Automation* (R. Dorf, ed.), pp. 308–322, New York, NY, USA: John Wiley and Sons, 1988.

[5] M. R. Cutkosky and J. M. Tenenbaum, "Providing computational support for concurrent engineering," *International Journal of Systems Automation: Research and Applications*, vol. 1, no. 3, pp. 239–261, 1991.