
Flexible Integration Model for Virtual Prototype Families

Stephan Vornholt* and
Ingolf Geist

Faculty of Computer Science,
University of Magdeburg,
Magdeburg, Germany
E-mail: {stephan.vornholt|ingolf.geist}@ovgu.de
*Corresponding author

Abstract: For many years computer-based concurrent engineering of products, denoted as Virtual Engineering (VE), has been developed to meet requirements in product development. Current research in concurrent engineering deals with the integration and synchronization of domain-specific (CAD design, simulation analysis) data and tools. This paper focuses on a flexible, multidimensional data model that integrates information and models from different engineering domains. The data model describes all information about a Virtual Product (VP) as *Virtual Prototype Family*. Operations for VP-family creation, management, and extension are defined. The proposed data model and manipulation operations allow a flexible connection and synchronization of domain specific tools within the design process as well as consistency control during the complete VE process. The data model is planned to be used in a VE system as the foundation of a metadata repository storing model information during the engineering process.

Keywords: Virtual Engineering, Integration, Virtual Prototype Families

1 Introduction

Product development requires the collaboration of several expert groups. Every group has its own point of view to the development process as well as to the product data, but all groups also share knowledge, information, and even active analysis models. Therefore, data integration and synchronization solutions are necessary to overcome the heterogeneity and to allow concurrent work between different groups. In this work we consider the process of *Virtual Engineering (VE)* which is defined in the phases design and analysis/simulation of the product lifecycle process. The following three engineering scenarios are considered:

1. An intermediate CAD model is constructed at first and corresponding simulation models (e.g., multi-body system (MBS), finite element (FE) model) are derived from the CAD model. Following, every domain model evolves independently. Dependencies to the original model as well as between simulation

models have to be managed to allow feedback between design and analysis models.

2. Given any conceptual model (formal or informal), all domain models are developed independently. At some point of the process the results have to be combined to evaluate the development success, e.g., tests of stress at different parts. Hence, a support for combining models is necessary.
3. As product development relies heavily on previous results that are modified or reused, design libraries or repositories are extremely important [16]. Nowadays, the libraries for each domain are mostly independent and connections between different domains have to be recreated in every development process. A combined management of design libraries is necessary to reuse defined connections.

The management of different models and especially their dependencies is required to support the previous scenarios. A data model allowing the description of these objects is the foundation of such a management. Although many approaches in the literature were proposed (see Section 2), we introduce a flexible model that describes the complete data of a *Virtual Prototype (VP)* in this work. It is based on following principles. The development of domain models according to a product describes a *Virtual Prototype Family*. A family contains variants and versions of the VP in three dimensions: *components*, *domain models*, and *instances*. Components are conceptual entities which describe a prototype. They can be hierarchically combined and communicate via interfaces. One component encapsulates all information to one entity. Domain models describe form, function, or behavior of the entity while an instance is a parametrized domain model, and is described by parameter value lists. Parameter value dependencies between instances can be resolved using domain models and component relationship descriptions.

Figure 1 outlines the proposed architecture and illustrates the context of this work in a product design management system. The metadata repository stores the conceptual structure of the VP. Every component, denoted as C , contains model descriptions (M) of different domains and the dependencies between them. The actual models are stored in domain specific databases which are organized by a PDM system. Changes of the domain models are sent to the metadata repository. Domain specific tools work on their own data but use additional metadata from the repository. This information is obtained via tool plug-ins or a management console. All metadata related data flows are implemented using Web services for taking into account system extensibility.

In the remaining paper we address the following problems: (i) the definition of a multidimensional data model for describing a VP-family. (ii) definition of explicit dependencies between domains and within domains. (iii) specification of operations to define, extend, and change families in a flexible way and to create views for each point of interest. In Section 2 related work is discussed. Section 3 defines the proposed multidimensional data model. Subsequently, operations on the model are discussed in Section 4. The paper concludes with a summary of the results and a discussion of further development steps in Section 5.

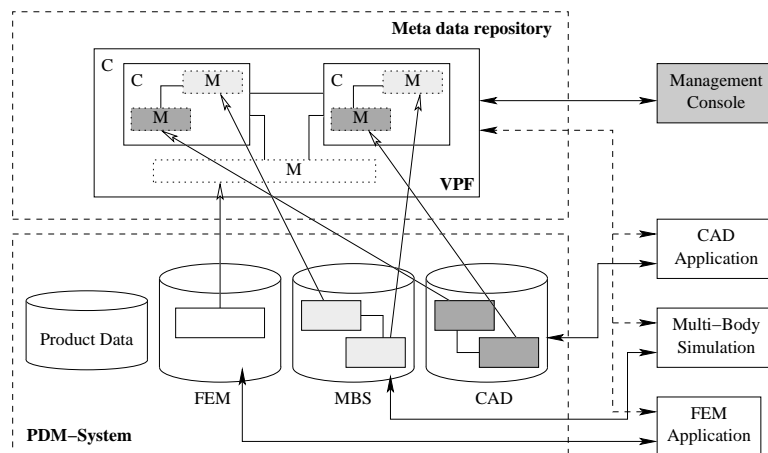


Figure 1 System Architecture

2 Related Work

Data integration solutions for product development processes are surveyed for instance in [7, 12, 15, 19]. Related approaches can be classified into:

Integration using product models: Product models have been created to express the specification of products in all facets. Examples are the Core Product Model (CPM) [4], the standard STEP [9], or the property-driven product model [18]. With extensions of these models (e.g., MECHASTEP) it is possible to integrate behavior, form, and function models from different domains into a Virtual Prototype. However, complex, universal product models are hard to implement and limit the local view of domain experts.

Automatic model transformation: Prototypes are created and parametrized preferably in one system, e.g., a CAD system. Subsequently, different simulation models (and instances) are derived for analysis, e.g., FE models [1]. Advantages are the tight integration of design and analysis as well as a single point of manipulation. The disadvantage is the missing flexibility caused by dependency from one product, and therefore the existence of unfamiliar tools for some expert groups.

Model-driven architecture: The Model-Driven Architecture (MDA) is a (software) engineering approach using models as well as corresponding data model transformation during product development. Extensions to UML [3, 5, 13] have been proposed supporting different kinds of models in mechatronic design processes. SysML [13] acts as an OMG standard. A comprehensive survey of visual model languages is given in [6]. MDA allows the definition of reusable software components in product design and can partly support VE. But there is still the open question of the integration of PDM, CAD on the one hand and CAE on the other hand.

Object-oriented modeling: Another closely related approach is object-oriented modeling of VP fragments. Fragments are hierarchically organized in components and can be reused as software components [10, 11, 14, 17]. Interfaces, composed by ports, hide implementation details. Furthermore, integration data models allow the specification of alternative models for one artifact, e.g., components. However,

the approaches lack of a formal way of combination of models within components. Additionally, instances and models are not separated.

Multi-aspect models: In multi-aspect models, all domain-specific information is stored using the domain data model, if possible a domain-specific standard model. The models are linked by a product master model [8] or a lightweight link model based on a metadata model [2] with the goal of update synchronizations, but combined libraries or coupling of combined components are not considered.

3 Model

3.1 Fundamental terms

A **Domain** is a self-contained part of an abstract or physical space where a Virtual Prototype can be viewed, modeled, and validated. A **Domain Model** (*DM*) describes the behavior, form, or function of a Virtual Prototype in one engineering domain. A domain model may be composed by sub-models. The basic building blocks of domain models are denoted as **Basic Model** (*BM*), and is not decomposable in the given context. Both kinds of models –domain and basic model– feature parameters for customization and ports as communication to other models of the same domain. Generally, we will speak of **Model** (*M*) when we mean a domain-specific VP description. For instance, a body model is a basic model in multi-body simulation (MBS), and the complete MBS system represents one domain model of a Virtual Prototype. An **Instance** is a fully parametrized object of type of a model, e.g., an actual body in a MBS system is parametrized with its mass and inertias. **Parameters** of a model are quantities that describe the characteristics of the model, and analogously, the characteristics of a VP in a certain domain. They are defined by a pair consisting of a name and a data type. A **Port** is a connection point where models can be combined. Through connected ports signals, material, and forces are transferred. For example, ports in a MBS system are points where forces are exchanged.

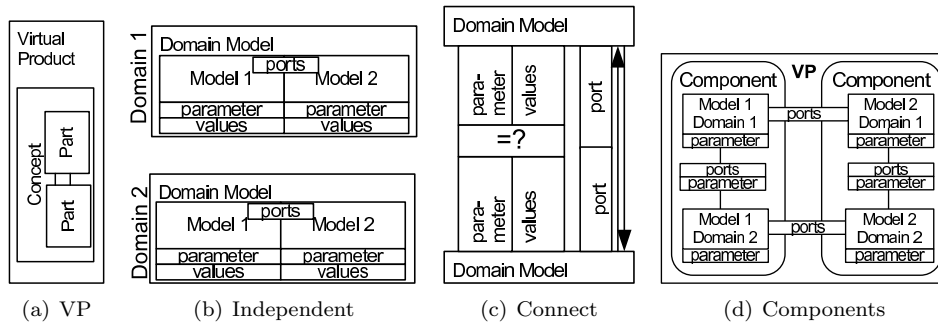


Figure 2 Domain interconnection

Figures 2(a-c) describe the connection between different domain models in an overview. Assuming a Virtual Prototype is conceptually structured into two sub-parts (Fig. 2(a)). Commonly, the VP is modeled and tested independently in different domains using domain-specific models and instances (Fig. 2(b)). In order to exchange parameter values or to synchronize results, the domain models have to

be connected (Fig. 2(c)). However, the original conceptual structure of the VP is lost, the different parts of the VP have to be tested in both domains together.

Therefore, the construct **Component** is introduced. A component represents a conceptual part of a Virtual Prototype and encapsulates all domain models as well as their dependencies for this part. A component ensures the dependencies between different domains, while domain models combine sub-models within one domain. A component comprises an interface which allows the parametrization of internal domain models and the combination of components to complex components. The result is a hierarchical model of the VP where inter-domain dependencies are also managed in building blocks. Fig.2(d) illustrates the example using components. The informal description of the parts of the integration model as an UML schema is given in Fig. 3.

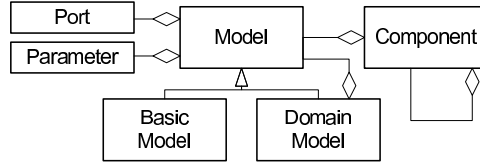


Figure 3 UML-Term-Schema

3.2 Formal definitions

Domain Model: A domain model is composed of models and their connections. A basic model is defined as a 3-tuple $BM = (id, \mathbf{P}, \mathbf{PM})$ with id as an identifier within a domain model, \mathbf{P} a set of ports, and \mathbf{PM} a set of parameters. Ports and parameters are defined as pairs (id, \mathcal{D}) with \mathcal{D} as the data type. The set $\mathcal{BM}(D)$ comprises then all basic models of a domain D . A domain model is a tuple $DM = (id, \mathbf{P}, \mathbf{PM}, \mathbf{M}, \mathbf{CN})$ with id as identifier, \mathbf{M} the set of contained models, and \mathbf{CN} as set of connections. $\mathcal{DM}(D)$ is the set of all domain models in D . The model set \mathbf{M} is a subset of $\mathcal{BM}(D) \cup \mathcal{DM}(D)$. The connection set $\mathbf{CN} \subset \mathbf{M.P} \times \mathbf{M.P} \times \mathbf{CS}$ describes the port connection between connected models, that are refined using a constraint $cs \in \mathbf{CS}$. The set $\mathbf{M.P}$ describes all pairs of models and contained ports, i.e., $\mathbf{M.P} = \bigcup_{m \in \mathbf{M}, p \in m.P} (m, p)$.

Component: A component is a conceptual part of a Virtual Prototype. Formally, a component is a tuple

$$C = (id, \mathbf{M}, \mathbf{C}, \mathbf{P}_{\text{DEP}}, \mathbf{PM}_{\text{DEP}}, \mathbf{I})$$

with id an identifier, set \mathbf{M} is a set of models, where each model is from a different domain. The set \mathbf{C} describes contained components. Furthermore, it contains \mathbf{P}_{DEP} representing possible port mappings between different domains and \mathbf{PM}_{DEP} as a set of parameter dependencies. The dependencies are triples $(\mathbf{P}'_1, \mathbf{P}'_2, cs)$ with $\mathbf{P}'_i, i = 1, 2$ are subsets of ports or parameters of one particular domain model and cs a constraint formula that describes the dependency of the ports or parameters, respectively. The interface $\mathbf{I} = (\mathbf{P}, \mathbf{PM}, \mathbf{MP})$ to a component, includes all external parameters and ports as well as a set of mappings $\mathbf{MP} = (\mathbf{I.P} \times (\mathbf{M.P} \cup \mathbf{C.I.P})) \cup (\mathbf{I.PM} \times (\mathbf{M.PM} \cup \mathbf{C.I.P}))$ which maps the interface to the internal ports and parameters of the domain models or sub-component interfaces, respectively.

Instances: Instances are (partially) parametrized objects of models and concepts. A concept instance can be described by a tuple $CI = (id, C, \mathbf{PV})$ with id an identifier, C the corresponding component, and \mathbf{PV} a set of parameter values, i.e., pairs of the form (pm, v) , $pm \in C.I$ and $v \in pm.D$. A component is also related to a set of domain model instances: $DI = (id, DM, \mathbf{PV})$.

Virtual Prototype Family: A Virtual Prototype family is the combination of all components, domain models, and instances. Hence, the family is defined as $\mathcal{F} = (\mathbf{C}, \mathbf{CI})$ with \mathbf{C} a set of connected components with corresponding models and \mathbf{CI} a set of connected instances with corresponding model instances.

3.3 *Case study*

A simple example of an industrial robot, denoted as RobotArm, illustrates the formal definition of Virtual Prototype families. Figure 4 illustrates two domain specific representations of a Virtual Prototype: a CAD model and a MBS model. Conceptually, the system consists of a base and two connected movable arms.

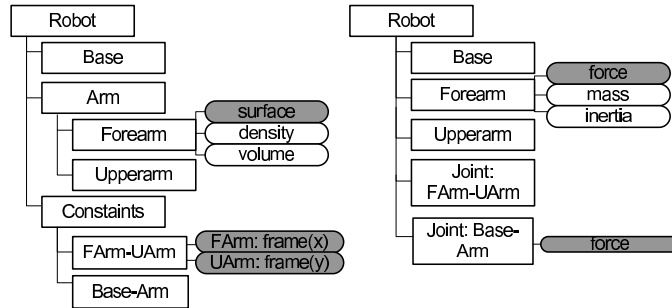


Figure 4 Two domain models of the case study

The corresponding VP family contains a set of components that represent either a rigid body or a connection between them in both models. Following this idea, a set of rigidly connected parts (CAD) and body models (MBS) are placed in particular components, and connection constraints in CAD and joints in MBS are put into separate components. In every component, the dependencies are locally defined and subsequently, only components have to be combined. Fig. 3.3 illustrates the decomposed structure according our model.

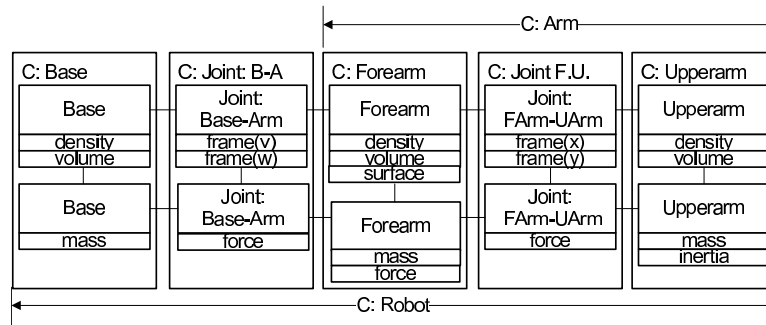


Figure 5 Component Structure of Robot Arm

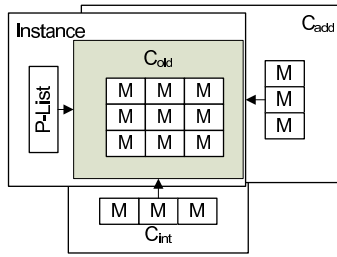


Figure 6 Manipulation Operations

Considering component *forearm* (Fig. 3.3) that includes one "part" model of the CAD domain. The part model is represented by a basic model

$$part = (part, \{(surface, IGES)\}, \{(dens, float), (vol, float)\}).$$

The model has one port described by the surface. Furthermore, two parameter exist: the density(*dens*) of the material as well as the volume (*vol*) of the part. An enclosing component is described as follows:

$$C = (forearm, \{part\}, \{\}, \{\}, I)$$

$$I = (\{(surface, IGES)\}, \{density, vol\}, MP)$$

$$MP = \{(I.surface, part.surface, =), (I.dens, part.dens, =), (I.vol, part.vol, =)\}$$

The example shows that ports and parameters are mapped to the component interface and the values are not modified in that case. Dependency sets are empty, because the component contains only one model.

Assuming now the integration of an MBS model in *C*, one dependency within one component between CAD and MBS models is the following. The part model offers the parameters volume (*part.vol*) and density (*part.dens*) and the MBS model has a parameter *body.mass*, the dependency is

$$cs_1 = (\{part.vol, part.dens\}, \{body.mass\}, \\ part.vol * part.dens \stackrel{==}{\Rightarrow} body.mass). \quad (1)$$

By using the metamodel components, once described dependencies can be reused in other designs.

4 Manipulation Operations

To build variants of VP-families it is reasonable to support manipulations (addition, deletion, and updates) of components. The integrity of components and their sub-components is a necessary precondition to enable recombination. Thus, the VP-family operations have to ensure consistent port and parameter mappings as well as interface mappings. Figure 6 shows the three dimensions of manipulation: integration of domain models (*C_{int}*), addition of components (*C_{add}*), and instantiation of components and models (*P-List*).

Domain operations: Addition of models, denoted as \otimes_{int} , integrates a new domain to a component. I.e., a new variant of the component is created. The integration operation is defined as 3-tuple $\otimes_{int} = (C, M, D)$ with model M , component C and a set of dependencies $D = \{P_\otimes\} \cup \{PM_\otimes\}$, where $P_\otimes \subset C.P \times M.P \times CS$, with CS the set of constraints, describes the port matches and $PM_\otimes \subset C.PM \times M.PM \times CS$ the parameter matches between model and component. C_{int} keeps the identifier id since the component is an extended version. While the new model M is appended to the set of component models \mathbf{M}_c , the dependency sets PM_{DEP_c} and ports P_{DEP_c} depend on the \otimes_{int} -operation where port and parameter dependencies and their constraints cs are defined. The dependencies and mappings affect the interface I_{int} because bound ports $P_{d_\otimes} \subset P_\otimes$ and derived parameters $PM_{d_\otimes} \subset PM_\otimes$ can be ignored. The model is integrated independently from the domain models connection set CN_m . Separating or deactivating a domain releases the bound ports. The integration of model $M = (id_m, P_m, PM_m, M_m, CN_m)$ into the component $C = (id_c, M_c, P_c, PM_{DEP_c}, I_c)$ with the matching ports P_\otimes and parameters PM_\otimes delivers the following component:

$$\begin{aligned} C_{int} &= (id_c, \mathbf{M}_c \cup \{M\}, P_{DEP_c} \cup P_\otimes, PM_{DEP_c} \cup PM_\otimes, I_{int}) \\ I_{int} &= I_c \cup P_m \cup PM_m \setminus (P_{d_\otimes} \cup PM_{d_\otimes}) \end{aligned}$$

The modified component cannot be decomposed since the model was integrated as a block. Considering the definition of models in Subsection 3.2 it is possible to decompose the DM and afterwards apply the operation \otimes_{int} to the part-components. Furthermore, a semi-automatic match-algorithm for ports and parameters using the constraints of $I.M$ and \otimes_{int} may be applied.

The operation is illustrated by extending the component `forearm` (see Sec. 3.3) by a body model of the MBS domain, i.e., the model $body = (body, \{\}, \{force\}, \{inertia, mass\})$. The operation is assumed as $\otimes_{int} = (forearm, body, \{I.vol, I.dens, M.mass, cs_1\})$ with cs_1 defined in Eq. (1). The result is the following modified component:

$$\begin{aligned} C &= (forearm, \{part, body\}, \{\}, \{cs_1\}, I) \\ I &= (\{surface, force\}, \{density, vol, inertia\}, MP) \\ MP &= \{(I.surface, part.surface, =), (I.force, body.force, =), (I.dens, part.dens, =), \\ &\quad (I.vol, part.vol, =), (I.inertia, body.inertia, =)\} \end{aligned}$$

Component operations: The combination of components creates a component containing them and a set of port and parameter dependencies. New connection information for ports P_\otimes and parameters PM_\otimes as well as a new interface I_{add} are created. The new interface ports and parameter depend on the constraints cs . Bounded ports can be ignored. The following equation shows the combination of two components of $C_1 = (id_1, M_1, P_{DEP_1}, PM_{DEP_1}, I_1)$ and $C_2 = (id_2, M_2, P_{DEP_2}, PM_{DEP_2}, I_2)$.

$$\begin{aligned} &\otimes_{add}(C_1, C_2, \{P_\otimes\} \cup \{PM_\otimes\}) : \\ C_{add} &= (id, \{\}, \{C_1, C_2\}, P_\otimes, PM_\otimes, (I_1 \cup I_2 \setminus (P_{d_\otimes} \cup PM_{d_\otimes}))) \end{aligned}$$

Now we are able to combine the components `forearm` and `joint:f.u.` with their respective models to component C_{arm} which contains a combined interface

for both:

$$C = (\text{joint:f.u.}, \{\text{connection}\}, \{\}, \{\}, (\{\text{frame}(x), \text{frame}(y), \text{force}\}, \{\}, MP))$$

$$\otimes_{add} = (\text{forearm}, \text{joint:f.u.}, \{(I.\text{frame}(x), I.\text{surface}), (I.\text{force}, I.\text{force})\})$$

$$C_{arm} = (\text{arm}, \{\}, \{\text{forearm}, \text{joint:f.u.}\}, \{cs_1\}, I)$$

$$I = (\{\text{surface}, \text{force}\}, \{\text{density}, \text{vol}, \text{inertia}\}, MP)$$

Updates: Updating an instance is done by adding a new parameter-value list to the chosen component interface, which distributes them to sub-components and models and creates instances. Substitutions or modifications of components or models which are part of a complex component trigger the creation or deletion of parameters, ports, and their corresponding mappings. While those operations can be mapped on the component and domain operations, missing ports have to be addressed. An update is well-formed, when it handles all previously bounded ports and parameters.

Instance operations: Instances define values for interface parameters. When all required parameters are initiated, each model of the component can create a valid executable domain model. Dependent or invisible parameters are automatically derived with help of a VP-family.

5 Conclusions

This paper presented a flexible, multidimensional integration model which stores model information of different engineering domains as well as their dependencies in enclosing components. The components describe the semantic structure of a Virtual Prototype and allow the reuse of dependencies between domains. The set of all variants and versions of components form a Virtual Prototype family. A presentation of a simplified case study illustrates the usability of the integration model. Furthermore, consistent manipulation operations as well as integration operations for domain models were defined and used to illustrate the flexibility of the model.

Previous considerations show that many possibilities have not been elaborated and new approaches will be needed. The described VP-family model will be the base of a more complex metadata repository, which is planned to monitor Virtual Engineering processes on the one hand and to provide metadata information to all process members and tools on the other hand.

Acknowledgments: The work described in this paper has been supported by the European Commission: European Regional Development Fund, project ‘‘COMO’’ C1-3201201 and C3-3201201.

References and Notes

- 1 C.G. Armstrong. Modeling Requirements for Finite-Element Analysis. *Computer-Aided Design*, 26(7):573 – 578, 1994.

- 2 Sabeur Bettaieb and Frdric Nol. A generic architecture to synchronise design models issued from heterogeneous business tools: towards more interoperability between design expertises. *Engineering with Computers*, 2007. online first.
- 3 Sven Burmester and Holger Giese. Visual Integration of UML 2.0 and Block Diagrams for Flexible Reconfiguration in MECHATRONIC UML. In *2005 IEEE VL/HCC 2005*, pages 109–116. IEEE Computer Society, 2005.
- 4 Steven J. Fenves, Sebti Foufou, Conrad Bock, Rachuri Sudarsan, Nicolas Bouillon, and Ram D. Sriram. CPM 2: A Revised Core Product Model for Representing Design Information. Technical Report NISTIR 7185, National Institute Of Standards and Technology, 2004.
- 5 Matthias Gehrke, Jan Meyer, and Wilhelm Schäfer. Modellierung von Softwarekomponenten für mechatronische Systeme in UML auf Basis von Systemstrukturen. In *5. Workshop "Entwurf mechatronischer Systeme"*, pages 145–156, Paderborn, 2007. Heinz Nixdorf Institut.
- 6 Holger Giese and Stefan Henkler. A survey of approaches for the visual model-driven development of next generation software-intensive systems. *Journal of Visual Languages and Computing*, 17:528 – 550, 2006.
- 7 Dave Hislop, Zoé Lacroix, and Gerald Moeller. Issues in mechanical engineering design management. *SIGMOD Record*, 33(2):135–138, 2004.
- 8 C.M. Hoffman and R. Joan-Arinyo. CAD and the Product Master Model. *Computer Aided Design*, 30(11):905 – 918, 1998.
- 9 ISO. *ISO 10303:1994 – Industrial Automation Systems and Integration - Product Data Representation and Exchange*, 1994.
- 10 Vei-Chung Liang and Christiaan J. J. Paredis. A Port Ontology for Conceptual Design of Systems. *Journal of Computing and Information Science in Engineering*, 4(3):206 – 217, 2004.
- 11 Gregory Mocko, Richard Malak, Christiaan Paredis, and Russell Peak. A Knowledge Repository for Behavioral Models in Engineering Design. In *ASME DETC 2004*. ASME, 2004.
- 12 Gregory M. Mocko and Steven J. Fenves. A Survey of Design – Analysis Integration Issues. Technical Report NISTIR 6996, National Institute Of Standards and Technology, 2003.
- 13 OMG. *OMG Systems Modeling Language (OMG SysML), V1.0 – OMG Available Specification*, September 2007. <http://www.omg.org/spec/SysML/1.0/PDF>.
- 14 C.J.J. Paredis, A. Diaz-Calderon, R. Sinha, and P.K. Khosla. Composable Models for Simulation-Based Design. *Engineering with Computers*, 17(2):112 – 128, 2001.
- 15 Joshua D. Summers, No Vargas-Hernndez, Zuozhi Zhao, Jami J. Shah, and Zo Lacroix. Comparative Study of Representation Structures for Modeling Function and Behavior of Mechanical Devices. In *ASME DETC 2000*. ASME, September 2001.
- 16 S. Szykman, R.D. Sriram, C. Bochenek, J.W. Racz, and J. Senfaute. Design repositories: engineering design’s new knowledge base. *Intelligent Systems and Their Applications, IEEE*, 15(3):48–55, May/June 2000.
- 17 Noe Vargas-Hernndez, Jami Shah, and Zo Lacroix. Knowledge Representation for Conceptual Engineering Design. In *ACIS SNPD 2002*. ACIS, June 2002.
- 18 Christian Weber, Horst Werner, and Till Deubel. A different view on Product Data Management/Product Life-Cycle Management and its future potentials. *Journal of Engineering Design*, 14(4):447 – 464, December 2003.
- 19 F. Zorriassatine, C. Wykes, R. Parkin, and N. Gindy. A survey of virtual prototyping techniques for mechanical product development. *Journal of Engineering Manufacture*, 217:513 – 530, 2003.