FÖDERAL: MANAGEMENT OF ENGINEERING DATA USING A SEMISTRUCTURED DATA MODEL

Christoph Mangold, Ralf Rantzau, Bernhard Mitschang
Universität Stuttgart
Universitätsstr. 38, 70569 Stuttgart
firstname.lastname@*informatik.uni-stuttgart.de*

Key words: product data management, semistructured data, integration, data modeling

Abstract: The Föderal system is a flexible repository for the management, integration and modeling of product data. Current systems in this domain employ object-oriented data models. Whereas this is adequate for the *management* of product data, it proves insufficient for integration and modeling. Present semistructured data models, however, are suited ideally for integration, but data management and also modeling is a problem. In this paper we describe our approach to narrow down the gap between structured and semistructured data models. We present the Föderal information system which employs a new semistructured data model and show how this model can be used in the context of management, integration, and modeling of engineering data.

1 Introduction

Today, the situation of engineering companies is determined by increasing time pressure. The efficient management of product data along the entire product life-cycle has become a prerequisite for companies to succeed on the market. Current product data management (PDM) systems claim to achieve this. The engineering companies cooperating in the Föderal project, however, regard these systems as not flexible enough, mainly because these systems use object-oriented data models to represent product data. To avoid the restrictions of object-oriented models we present the Föderal[1] Information System for product data management featuring a semistructured data model.

1.1 Semistructured Data

Recently, semistructured data models and their applications have attracted attention and a lot of research is devoted to this subject, e.g. (Abiteboul et al., 2000; Buneman, 1997; Garcia-Molina et al., 1997; Goldman and Widom, 1997; W3C, 2002; Suciu, 1998; Quass et al., 1995). In structured data models, schema and data are clearly separated and the data always conforms to the schema. This conformity check is done automatically upon creation and update of data. This is not the case for semistructured data (Buneman, 1997; Suciu, 1998; Abiteboul et al., 2000). First, it is possible that there is no schema information available at all. Second, if there exists a separate schema then conformity checks always have to be invoked explicitly.

1.2 Product Data Management Systems

The domain of product data management (PDM) systems is one of the most important in today's manufacturing industry. During all phases of the product life-cycle data is produced or consumed by various applications. Therefore, the centralized management of product data has become crucial.

Current PDM systems can be considered *repositories* (in terms of (Bernstein and Dayal, 1994)) for engineering data. They provide a domain specific information model which includes *meta data* describing the data in the vault, as well as *services*. Typical services of both, repositories and PDM systems are, e.g., check-in/check-out, version control, and configuration control. In most cases the information model of a PDM system offers some predefined base classes.

[1]The Föderal project is supported by the German Federal Ministry of Education and Research (BMBF). The project web site can be found at `www.foederal.org`. "Föderal" is German for "federated".

These classes have to be extended and supplemented in a customization process to better support company-specific applications and workflows.

To the best of our knowledge and according to (Schöttner, 1999; Abramovici and Sieg, 2001), all current PDM systems use object-oriented information models. This includes commercial systems like Metaphase and Enovia, research prototypes like the IDEE system (Schönhoff et al., 1997) and the Product and Production Model (PPM)-based system in (Hillebrand et al., 1998), and the OMG project PDM Enablers (OMG, 2001).

1.3 The Föderal Project

In the Föderal project, three internationally recognized mechanical engineering companies cooperate in identifying their problems in product data management, investigating appropriate solutions, and realizing an information system to accomplish optimal information supply.

It turned out that they not only need a standard product data *management* system. As we will explain in detail in Section 2, they additionally focus on two further issues. First, the *seamless integration* of legacy systems and legacy data into the new system is crucial. Second, the companies involved in the Föderal project design machines according to a *specific modeling methodology* which has to be supported by IT infrastructure. To achieve these goals, we designed the Föderal Information System (FödIS). Basically, FödIS is a federated information system to integrate legacy database systems. The federated schema of FödIS is based on the data model which will be described in Section 3.

1.4 Outline of the Paper

The remainder of the paper is organized as follows: In Section 2, we argue why object-oriented data models have severe deficits when employed in our scenario and what we gain by using a semistructured model. In Section 3, we show the shortcomings of current semistructured data models and describe our proposal for a new data model. We give a report on the system development status in Section 4 and conclude the paper in Section 5.

2 The Case for a Semistructured Data Model

Current PDM systems employ object-oriented data models. However, they are neither suited to enable an iterative modeling process, nor to support the integration of heterogeneous component systems. In this section, we explain why these two tasks are crucial to any PDM system in the application area of mechanical engineering.

2.1 Modeling

To speed up and simplify the product development process, the companies cooperating in the Föderal project have developed a module-based design methodology. Typically, new machines are created as aggregations of parameterized modules. A module is an assembly of components like software, hardware, documentation, etc. For example, a module describing a spindle, as sketched in Figure 1, consists of CAD data, circuit diagrams, various text documents for development, maintenance and service guidelines, and programmable logic controller (PLC) code to control the movements of the spindle.

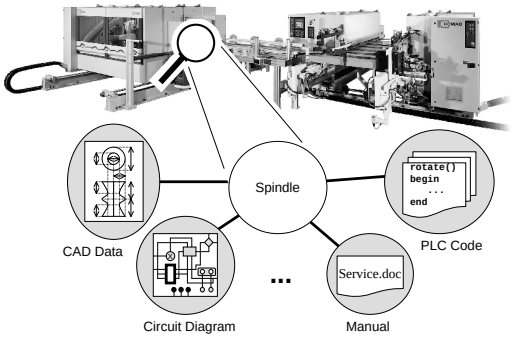

Figure 1: A spindle module and its components in a double-end tenoner machine (Homag Optimat).

Modules comply to certain restrictions which make them usable as building blocks in higher-level modules in the design process of new machines. They are developed in an application-centered manner: Often a new machine cannot be built entirely from modules alone, but it has to be supplemented with non-standard parts. Non-standard parts do not necessarily comply to the restrictions of modules, i.e., they are not reusable. If a non-standard part turns out to be useful in later projects, then it might be upgraded to fulfill the requirements of modules and thus be made available for future use. It is important to note that modules are not designed from scratch. Rather some non-standard parts eventually turn out to be useful. In this case, these parts become new modules.

The companies cooperating in the Föderal project have in common that rarely any two machines they manufacture are exactly alike. Thus, the processes of developing non-standard parts and declaring new modules are common procedures and are performed frequently. For example, a new customer requires the

engineering company to develop an extremely fast-rotating spindle to raise the throughput of a new machine. This fast-rotating spindle is then developed to fit into this specific machine only, i.e. parameters, error messages, documentation, etc. are hard-coded. Later on the fast-rotating spindle turns out to be a common requirement. Then the company decides to build a module from this non-standard part.

Object-oriented data models, however, do not support this kind of methodology. In an object-oriented model, data always has to conform to the schema. This, however, is not useful to the engineering process of machines since irregular data occurs quite often in this domain. Hence, ordinary object-oriented data models are not flexible enough to support this well-approved methodology.

When semistructured data models were not popular yet, there were approaches to support this modeling process by adding flexibility to object-oriented data models. One of these approaches is the class-less object-oriented data model presented in (Groß-Hardt and Vossen, 1993).

2.2 Integration

The integration of legacy data into a federated schema is a crucial requirement for the Föderal system. On the one hand, this is necessary for the support and maintenance of systems in use. On the other hand, as we have explained in the previous section, new machines are built as a composite of modules. Hence the capital of the companies lies in the possibility to reuse modules and assemblies of machines they have already built and that have already proven to be of industrial strength, i.e., they are reliable, economical, etc. Thus, it is utterly important for a company to retain control of legacy data.

Today, all phases of the life-cycle of engineering products are supported by a diversity of so-called CAx tools. Usually, each of these tools embodies a proprietary data format and data management system. Hence, the different information sources that are responsible for the different phases in the engineering process of a machine are extremely heterogeneous. The variety of tools and data formats inevitably leads to incomplete or even contradictory data.

The integration of heterogeneous information sources is central to the management of product data. Present PDM systems try to solve this problem by using object-oriented data models. We observe, however, that semistructured data models like OEM (Garcia-Molina et al., 1997), YAT (Cluet et al., 1998) or XML (W3C, 2002) are superior in the domain of the integration of heterogeneous information systems. In projects like TSIMMIS (Garcia-Molina et al., 1997) and Xyleme (Reynaud et al., 2001) semistructured data models are used especially for the integration of incomplete and contradictory information coming from data sources that are not necessarily managed by a DBMS.

Generally speaking, the problem covered by projects like TSIMMIS and Xyleme matches the data integration problem of the Föderal system. To support the integration of heterogeneous and incomplete data, we follow the ideas presented in the scope of these projects by employing a semistructured data model in the federated schema of the FödIS.

2.3 Drawbacks

Of course, the realization of a federated schema based on a semistructured data model comes at some cost. Compared to structured models, semistructured models generally suffer from three main drawbacks:

Queries: From a semantical viewpoint, applications that use complex queries are hard to implement on top of a semistructured data model since the structure of the data can be changed and there is no mechanism to ensure the semantic "stability" of a query during changes of the data model. In our case this problem is not severe since the applications on top of the FödIS do not rely on complex queries. They mostly require navigational functionality. Nonetheless, we implemented a query language for our data model to facilitate simple (i.e. navigational) queries.

Performance optimization: Whereas it is a problem to query an evolving data model from a semantical point of view, it is even harder to optimize queries. Unlike in structured data models we cannot rely on stable indexes to speed up queries. As we explained above, however, our application scenario is not based on complex queries.

Human understanding: In semistructured data models not only the processing of queries is affected by the lack of structure but also user understanding can be derogated. Consider, e.g., a large OEM graph with thousands of edges. For a human user it will be much harder to extract information from this data graph than querying a relational data base.

Many solutions have been proposed to solve the problem of human understanding, e.g. the DataGuide (Goldman and Widom, 1997) for OEM, YAT (Cluet et al., 1998), and many other XML schema languages (see (Lee and Chu, 2000) for an overview). They all propose schemas for semistructured data. These solutions can be divided into two categories:

- Data to schema. Here, schemas are automatically derived from existing data. This approach focuses on the problem of query processing. However, a generated schema is not necessarily helpful to a human user.
- Schema to data. Several schema languages have

been developed, in particular for XML, that facilitate user understanding. Typically these schemas are somewhat less rigid, i.e., less precise than object-oriented schemas, although their expressive power increases steadily (Lee and Chu, 2000).

To support the module-based engineering methodology described in Section 2.1, human understanding of modeled data is crucial in our application domain. Hence we follow the *schema to data* approach of a manually defined schema and conformity checks.

3 The Föderal Data Model

In Section 2, we motivated our decision not to implement an object-oriented but to choose a semistructured data model. In this section, we present the Föderal Data Model (FöDM) as an extension of OEM. First, we argue why it was not an option to use pure OEM or any other existing semistructured data model we know of. Second, we show our requirements to design a new data model. After the specification of FöDM we describe two application scenarios. Finally, we compare FöDM with related data models.

3.1 Motivation to Extend OEM

Before designing FöDM we analyzed current semistructured data models. In particular we considered XML (supported by XML Schema), OEM and YAT. As explained above, our application area requires a flexible data model, which is well-suited for human interaction. The result of our investigations showed that the models fulfill our flexibility requirements. None of them, however, provides sufficient support for human interaction.

Instead of designing a new data model from scratch, we decided to extend an existing data model. We did not choose XML as a base model for two reasons: First, there is always a certain hierarchy induced by XML's object nesting relationship. This hierarchy is a restriction to the engineering methodology described above since real world scenarios rarely fit into one single hierarchy. Second, XML is not convenient to be handled by human users. This is true even in the presence of modern end-user modeling tools for XML. We do not see how either of these two issues could be ameliorated.

The distinct feature of the YAT data model is a particular mechanism for instantiation. Since this kind of instantiation is not needed in our domain, YAT is not a good basis for our model.

On the contrary, we considered OEM as a good basis for the following reasons: We think of OEM as a minimal but universal data model. It is extremely flexible since data and meta data can be part of the same OEM graph. However, human interaction with large OEM graphs is tedious.

3.2 Requirements

In this section, we will state our requirements to an extension of OEM with respect to human interaction and modeling. In particular we want FöDM to realize the following three features:

Bidirectional navigation of relationships: To enable an intuitive navigation in FöDM, edges are designed to be traversable in either forward or backward direction. This property, however, does not derogate the fact that edges are semantically directed. To consider relationships always as bidirectional pathways is a feature we already know from knowledge representation systems (Lenat and Guha, 1989).

Labels on nodes and edges: In the original OEM, data is represented as a graph with labeled nodes (Papakonstantinou et al., 1995). Subsequently a new notion of OEM has been introduced where labels are on edges instead of nodes (Abiteboul et al., 1997). XML can be seen as a node labeled graph. FöDM, in contrast, is a graph with labeled nodes *and* labeled edges. We introduced this property for the following reason: Representations of well-known data models like entity-relationship diagrams or UML class diagrams show that labels on both nodes and edges are needed to support the intuitive modeling process for human model designers. Furthermore, using labels on both, nodes and edges, leads to a very intuitive modeling of name-value-pairs. As a matter of fact, in a predecessor project of Föderal, which focused on modeling of engineering data, a semistructured data model with labeled nodes and edges proved useful (Dreyer et al., 2000).

Edge types: We introduce the concept of edge types to our model to provide a dynamic catalogue of all edge labels in the model. One reason for this is that FöDM is used not only by human users but also by algorithms. Since edge labels denote the relationship between nodes, algorithms will navigate the data model using edge labels, primarily. The second reason for introducing edge types is to build a semantic hierarchy among edge labels. The feature of edge types is well known from ontology languages such as DAML+OIL (Horrocks, 2002).

3.3 Specification of the Föderal Data Model

From the requirements discussed in Section 3.2 we equip FöDM with three different kinds of objects: nodes, edges and edge types.

A node object is specified as

Node: $\{$BasicType $value$, Edge[ ] in, Edge[ ] $out\}$

where BasicType is a simple data type like `integer` or `string`. It specifies the type of the *value* attribute of the node. Two nodes are connected via an edge object. The *in* and *out* attribute of a node object denote an array of incoming and outgoing edges, respectively. Note that the *in* attribute is needed to realize navigation in the opposite direction of the logical direction of an edge.

Edge objects are defined as follows:

$$\text{Edge: } \{\text{Node } source, \text{Node } drain, \text{EdgeType } type\}$$

The *source* and *drain* attributes of an edge denote the nodes connected by the edge. Again, we need the *source* attribute to navigate against the logical direction of an edge.

The label or the *type* of an edge is determined by its edge type which is defined as

$$\text{EdgeType: } \{\text{String } name, \text{EdgeType } superType\}$$

where *name* is the description of the edge type (i.e. the label of all edges referring to this edge type). The attribute *superType* is a pointer to any other edge type from which this edge type is semantically derived.

3.4 Example Scenarios

In the following, we illustrate the usage of our model by two examples. In the first example, we show how FöDM can be used to model engineering data in an intuitive and human understandable manner. In the second example, we present one aspect of data integration namely the integration of data from file systems.

Example 1: Data Model of a Spindle: In Figure 2, we illustrate the usage of our data model by showing a partial model of the spindle, described in Section 2.1. Different arrowheads represent different edge types. In the upper part, a Machine Unit, which is a component of a Machine, has four types of components: CAD-File, Circuit Diagram, Documentation, and PLC-Program. The Spindle is a specific kind of Machine Unit. To emphasize that these nodes are part of the model, they are connected to the Meta Data Model node by edges of type abstraction layer, cf. (OMG, 2000).

In the lower part, the Spindle instance S_BD4C is depicted with its actual components. In this example, S_BD4C has seven components which are instances of CAD-File, Circuit Diagram, Documentation, and PLC-Program, respectively. All instances in our example are connected to the Data Model node by an edge of type abstraction layer, cf. (OMG, 2000).

Notice that this is where our semistructured data model pays off: On the one hand, we can distinguish between meta data (i.e. classes) and data (i.e. instances) during all phases of the modeling process.

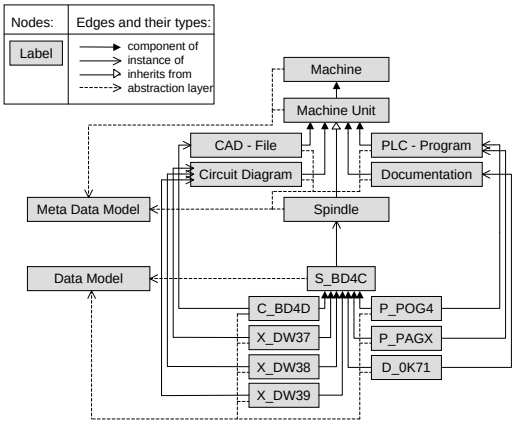

Figure 2: Partial data model of a spindle.

This also includes the possibility of structural on-demand consistency checks based on the edges of type abstraction layer. On the other hand, we are free to violate constraints in both, the data model and the meta data model at any time. Clearly, the design methodology described in Section 2.1 benefits from this characteristic, since it facilitates the usage of irregular data.

Example 2: Integration of data: In the application area of Föderal it is important to be able to access legacy data in a flexible way. In fact, most data produced by CAx tools resides in file systems. Therefore we will show how data from file systems can be accessed using FöDM. Figure 3 shows an example data model. On the left hand side there is the PLC program P_POG4 from the example above. We do not show the nodes for meta data model and data model from Figure 2 in this example. Furthermore, instead of introducing even more arrowheads, edge types are shown as labels next to the according edges. Consider the scenario where the source code for this program has to be retrieved by an application. Knowing the appropriate edge types the application will use the file transfer protocol (ftp) to copy the file /fodis/data/resources/P_POG4.plc from server FileServ to a local directory and start the application which can be found at C:\CoDeSys.exe with this file. If the user updates and closes the file, it will be copied back using the same information as for retrieving the file.

To integrate data from relational database management systems, nodes contain SQL queries instead of paths and filenames. These queries are executed dynamically to retrieve data from the respective system.

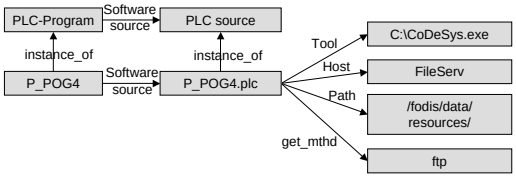

Figure 3: Integration of files using FöDM.

3.5 Related Models

To contrast FöDM with the most prominent related models we illustrate how to represent a small scenario in each model. Then we explain the differences, which are summarized in the table below.

To clarify the difference between FöDM and related models we choose the model of ODMG (Cattell, 1994) as the first reference model. The ODMG model is not semistructured, however, we feel that human understandability of this model is very good. Second, we chose OEM to show how our extensions affect the model. Finally, we illustrate the model in XML together with XML Schema (W3C, 2002).

We did not chose any model from the area of knowledge representation. We feel that the focus of knowledge representation systems differs from our application area in two significant ways. First, knowledge representation systems are primarily concerned with meta data. In contrast, we consider data and meta data equally important. Second, we do not want to extract unknown facts from our data, i.e. we do not need inference algorithms.

To avoid ambiguities, we will use the term *object* when we refer to a real world entity. Objects are interconnected by *relationships*. Abstractions of objects are called *object classes*. When we speak of *data* we refer to objects and relationships. A set of object classes is termed *meta data*.

In Figure 4(a), we show an extract of the FöDM example in Figure 3. While PLC-Program and PLC source represent object classes, P_POG4 and P_POG4.plc refer to objects.

For the ODMG example in Figure 4(b), we chose the UML notation to represent the class diagram (i.e. the meta data model) on top and the object diagram (i.e. the data model) right below. By modeling the two relationships implements and is_implemented_by we demonstrate how bidirectional navigation can be achieved in the ODMG model.

In Figure 4(c) we show the same example using OEM. Again, we introduce two edges per relationship to enable bidirectional navigation. In OEM nodes are not labeled except for the leaf nodes where the actual data is stored. Analogous to FöDM, data and meta data is stored in the same model.

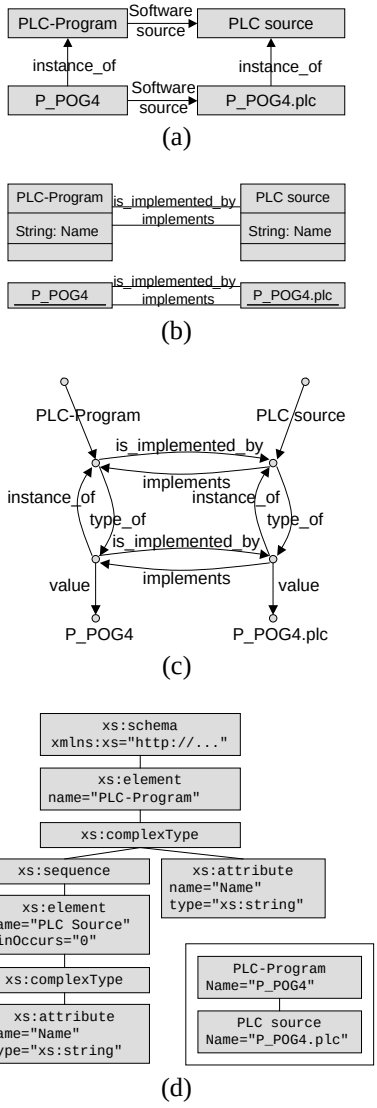

Figure 4: FöDM and related Models. (a) FöDM, (b) ODMG model, (c) OEM, (d) XML (inside the box) and XML Schema.

We show an XML example in Figure 4(d) using XML Schema to define the object classes. We are aware that this is not the only possibility to model the given scenario. Each gray box denotes an XML element with its name on top and attributes below. The two elements inside the rectangle represent the XML data file. The other elements belong to the XML Schema definition. The distinction between data and meta data in XML Schema is different from the ODMG model. In the ODMG model, data and meta data is clearly divided into class diagram and object diagram. The XML document, however, not only contains data but also meta data which is enclosed in

the identifiers of elements (e.g. PLC-Program).

The comparison between the respective data models reveals that FöDM offers the most intuitive way for human users to transfer real world scenarios to the data model. Even in the toy model of Figure 4, it is apparent that FöDM supports a lucid way to describe data. This pays off in real-life scenarios where models incorporate several thousand objects.

In Table 1 we summarize the differences of the data models we compared above with respect to the distinct features in FöDM.

Feature	FöDM	ODMG	OEM	XML
Labeled objects	+	+	–	+
Labeled relationships	+	+	+	0
Typed relationships	+	0	–	0
Bidirectional navigation of relationships	+	0	0	0
Separation of data and meta data	–	+	–	+

Table 1: Distinct features of FöDM compared to the models of ODMG, OEM and XML. The presence of a feature, the presence of related features or the possibility to simulate the feature and the absence of a feature in the respective model is expressed by +, 0, and –, respectively.

Labeled objects: A data model has labeled objects if objects can have names. There are labeled nodes in all models but OEM.

Labeled relationships: As explained above, there are also labels for edges in FöDM, which are realized by means of edge types. In the ODMG model we also encounter labeled relationships. Two objects are interconnected by pointers that are labeled with an identifier. In XML Schema there are no relationship labels. However, they could be simulated by XPointers or attributes of type IDREF. In OEM, in contrast, all semantic information is kept in labels on the edges of the data graph.

Typed relationships: The typing of relationships as described above is only available in FöDM. In the ODMG model and in OEM, names of relationships between objects can be restricted to a certain extent using the concepts of interfaces and derivation. Typing of relationships in FöDM, however, does not regulate the internal structure of objects but the structure among objects. In XML relationships between objects can be expressed using attributes, IDREFs, or the element nesting relationship. The difference between these relationship "types", however, is of a structural and not of a semantic nature. Furthermore, these types are not extensible as they are in FöDM.

Bidirectional navigation of relationships: FöDM is the only model which supports the bidirectional navigation of relationships. In the ODMG model and in OEM, this could be achieved by modeling each relationship twice, i.e. once for every direction. The hierarchical relationship in XML obviously can be navigated in either direction, but this does neither hold for relationships established by IDREF nor for XPointer or XLink attributes.

Separation of data and meta data: While there is a clear distinction between data and meta data in the object-oriented data model (which is the distinction between dynamic and static), there is no such distinction in FöDM (although it can be introduced artificially, as shown above). In XML, however, there is a clear separation between meta data which is the schema and the tags on the one hand, and data which is the content of the elements and attributes on the other hand. As in FöDM, the separation of data and meta data in OEM is not supported.

4 System Status

During the first phase of our project, the FödIS prototype has been implemented that incorporates FöDM based on a commercial DBMS.

Subsequently, the system has been introduced to one of the companies participating in the project and a model for their data has been established.

After migrating some company-specific programs to work on top of FöDM, the company has started designing the first machines using the above described methodology.

We also implemented a query language for FöDM to support the implementation of services and applications.

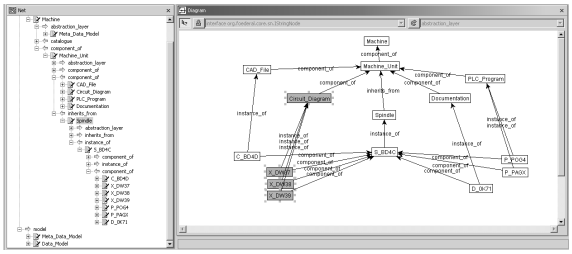

Figure 5: Two windows of the FödIS graphical user interface.

Figure 5 shows a portion of the graphical user interface of FödIS. It is implemented using the Eclipse framework for the Java programming language. On

the left side we see a tree representation of the data model of the spindle from Figure 2. On the right hand side a graphical representation of the data is depicted.

5 Conclusion

In this paper, we presented a new semistructured data model (called FöDM), that is used in the federated schema of our product data management system FödIS. We have learned from our industry project partners that future product data management systems will have to extend their focus to the modeling and integration of information. With FödIS we take a first step to fulfill these requirements. We have argued that object-oriented data models are neither flexible enough to support the modeling methodology prevalent in many mechanical engineering companies, nor to enable the smooth integration of legacy systems. Present semistructured data models foster integration of legacy systems, but they are not designed to support manual data modeling. For this reason we proposed a new semistructured model that emphasizes both, modeling and integration of information.

Acknowledgements

We thank our project partners at mind8, ISW, Homag, Nagel and Schuler for fruitful discussions, implementation, and for showing us the engineering point of view. We also thank the PFT at Forschungszentrum Karlsruhe for successful cooperation and BMBF for funding of the project.

REFERENCES

Abiteboul, S., Buneman, P., and Suciu, D. (2000). *Data on the Web - From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers.

Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. L. (1997). The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88.

Abramovici, M. and Sieg, O. C. (2001). PDM-Technologie im Wandel – Stand und Entwicklungsperspektiven. *Industrie Management*, 2001(5):71–75.

Bernstein, P. A. and Dayal, U. (1994). An overview of repository technology. In *VLDB'94*, pages 705–713. Morgan Kaufmann.

Buneman, P. (1997). Semistructured data. In *PODS'97*, pages 117–121. ACM Press.

Cattell, R. G. G., editor (1994). *The Object database standard, ODMG-93*. Morgan Kaufmann Publishers.

Cluet, S., Delobel, C., Siméon, J., and Smaga, K. (1998). Your mediators need data conversion! In *SIGMOD'98*, pages 177–188. ACM Press.

Dreyer, J., Lewek, J., and Angerbauer, R. (2000). Software-Architektur zur flexiblen Unterstützung von baukastenorientierten Entwicklungsprozessen. In *Tagungsband zum IT & Automation Kongreß Nürnberg*.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J. D., Vassalos, V., and Widom, J. (1997). The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132.

Goldman, R. and Widom, J. (1997). Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB'97*, pages 436–445. Morgan Kaufmann.

Groß-Hardt, M. and Vossen, G. (1993). Zur Entwicklung eines klassenlosen Objekt-Modells. In *BTW'93*, pages 306–315. Springer.

Hillebrand, G., Krakowski, P., Lockemann, P. C., and Posselt, D. (1998). Integration-based cooperation in concurrent engineering. In *IEDOC Workshop*, pages 344–355. IEEE Computer Society.

Horrocks, I. (2002). DAML+OIL: A reason-able web ontology language. In *EDBT'02*, volume 2287 of *Lecture Notes in Computer Science*, pages 2–13. Springer.

Lee, D. and Chu, W. W. (2000). Comparative analysis of six XML schema languages. *SIGMOD Record*, 29(3):76–87.

Lenat, D. B. and Guha, R. V. (1989). *Building large Knowledge-based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley Publishing Company.

OMG (2000). Meta Object Facility (MOF) specification. Available: http://www.omg.org/cgi-bin/doc?formal/2000-04-03 [1.6.2001].

OMG (2001). PDM enablers. Available: http://www.omg.org/homepages/mfg/mfgppepdm.htm [17.12.2001].

Papakonstantinou, Y., Garcia-Molina, H., and Widom, J. (1995). Object exchange across heterogeneous information sources. In *ICDE'95*, pages 251–260. IEEE Computer Society.

Quass, D., Rajaraman, Sagiv, Y., Ullman, J., and Widom, J. (1995). Querying semistructured heterogeneous information. In *DOOD'95*, volume 1013, pages 319–344. Springer.

Reynaud, C., Sirot, J.-P., and Vodislav, D. (2001). Semantic integration of XML heterogeneous data sources. In *IDEAS'01*, pages 199–208. IEEE Computer Society.

Schönhoff, M., Strässler, M., and Dittrich, K. R. (1997). Data integration in engineering environments. In *EFDBS'97*, pages 45–56.

Schöttner, J. (1999). *Produktdatenmanagement in der Fertigungsindustrie*. Carl Hanser Verlag München Wien.

Suciu, D. (1998). An overview of semistructured data. *SIGACT News*, 29(4):28–38.

W3C (2002). Available: http://www.w3.org/ [5.7.2002].