INTERNATIONAL DESIGN CONFERENCE - DESIGN 2004
Dubrovnik, May 18 - 21, 2004.

DESIGN 2004

A SOFTWARE BASED SYSTEM TO SUPPORT THE DESIGN FOR DISASSEMBLY

B. Gries and L. Blessing

Keywords: design for disassembly, CORBA, CAD, product models, software engineering

1. Motivation

Since 1994, the Collaborative Research Center Sfb 281 at the Technical University Berlin has been developing technologies, tools and methods needed to realize the vision of "Disassembly Factories for the Recovery of Resources from Product- and Material Cycles" [CRC, 2003]. Meanwhile, a number of different software tools have emerged – either with the purpose or the potential – to support the designer in creating disassembly- and recycling-friendly products.
During this process, the designer defines the materials, structure and connections of a product, aspects critical to disassembly and recycling [Pahl & Beitz, 1996]. Although tools that deal with each of these aspects exist, usually each represents an individual solution with limited possibilities to exchange data with other applications.
Looking up a material from a materials database, defining a material with the same properties in the CAD application and assigning the self-defined material to a part is therefore a common scenario. The efficiency of such practice, however, becomes questionable e.g. when a product is undergoing a Life Cycle Assessment (LCA) in order to evaluate its environmental impacts [Dose et al., 2003]. The retrieval of the material quantities within the product is a process that requires considerable manual effort due to the lacking ability of available LCA software to access the product data of existing CAD applications.
This example is somewhat typical for a general problem in CAD: the exchange of product data between applications that are based on different product models.

2. Partial product model

The software based system presented in this paper, called "Design Support System", is based on the concept that a common product model is the prerequisite for a successful integration of different development tools. Whereas an "integrated product model" should be able to describe all data within a product life cycle [Grabowski et al., 1993], a partial product model represents a finite set of product information, i.e. certain product aspects as part of another product model.

2.1 Requirements for a disassembly- and recycling-oriented partial product model

According to the abovementioned definition, a disassembly- and recycling-oriented partial product model should represent all information relevant to this phase of the product life cycle. For a decision on the *recycling strategy* of a product for example, knowledge of the *material* of each part is needed. Also, in order to perform a LCA, a part's *manufacturing process* must be known to acquire the necessary amount of energy.

The *product structure* plays a central role. It describes how the product components (i.e. assemblies and parts) are connected to each other using which *connection / fastening technology* and which components represent an assembly. Especially for non-destructive disassembly, it must be possible to derive a disassembly sequence from the partial product model. For destructive disassembly however, it is useful to define a cutting line along a part's surface.

2.2 Model specification

Figure 1 shows an UML class diagram of a partial product model that considers all of the aspects described in the previous section, though not all will be dealt with in this paper. The "Unified Modelling Language" is a graphical description method for the specification, visualisation and documentation of object oriented software systems developed by the "Object Management Group" (OMG). For a better understanding of Figure 1, only the most important elements of UML class diagrams are described in this paper. Detailed reference can be obtained e.g. from [Ambler, 2002].

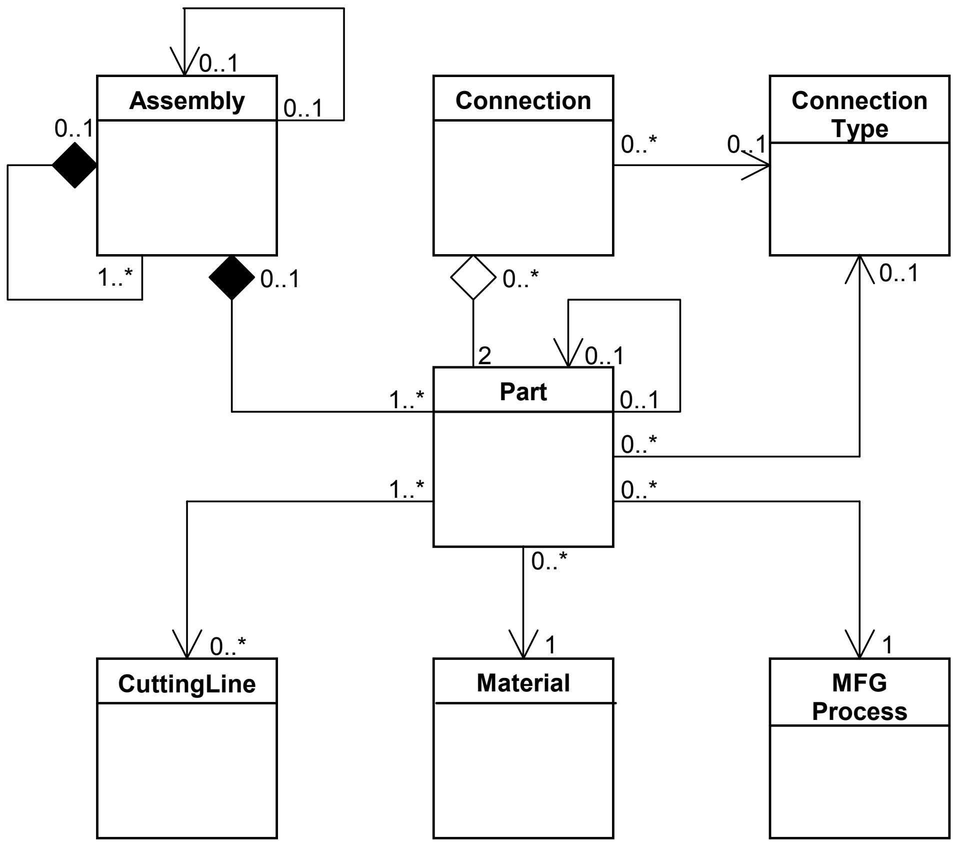


Figure 1. UML class diagram of the partial product model (properties and methods not displayed)

Each class is displayed as a box divided in three sections. The upper section contains the class name, the middle section the class properties and the lower section the class methods (properties and methods omitted in Figure 1). Associations are displayed as lines between the classes. Numbers at each end of a line describe multiplicity, meaning e.g. "1" exactly one, "0..1" zero or one, "1..*" one or more, and so on. Directionality is marked by an arrow head. Diamonds indicate aggregation, where filled diamonds designate a composition, e.g. an object physically containing another object.
Therefore, each *Part* can be contained by zero or one *Assembly* (0..1) while each assembly can hold any number of parts but at least one (1..*). In the same way, each assembly can be part of another assembly. In this model, each part consists of exactly one *Material*; the directionality expresses that it is possible to derive a material from a part, but not the other way around.

The product structure (as defined in 2.1) is modelled in the following way: each *Connection* is associated with exactly two parts and optionally with one connection type. In the latter case, the *ConnectionType* describes a joining method (e.g. gluing, welding, force fitting, etc.). A *Connection*-object not referencing a *ConnectionType*-object, indicates that there is at least one *Part*-object acting as a fastener. Screws, bolts, hose clamps etc. are treated as parts with the difference that they reference a connection type. By assigning each assembly and part a disassembly predecessor, it is possible to automatically generate a disassembly sequence for the whole product.

2.3 Modelling examples

Figure 2 and Figure 3 show two different modelling examples. The UML object diagram that is used is quite similar to a class diagram. Just as an object is an instance of a class, an object diagram could be regarded as an instance of a class diagram. Therefore, object diagrams describe the static structure of a system at a particular time. Again, each object is represented as a rectangle, which contains the name of the object and its class underlined and separated by a colon.

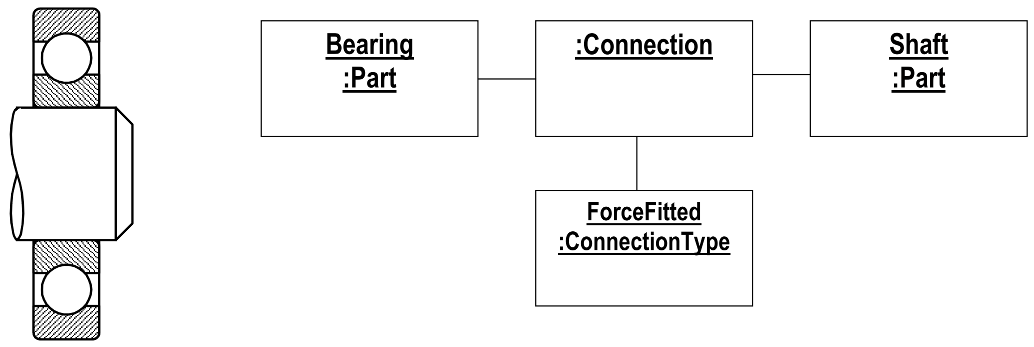


Figure 2. Simplified UML object diagram of a bearing force fitted to a shaft

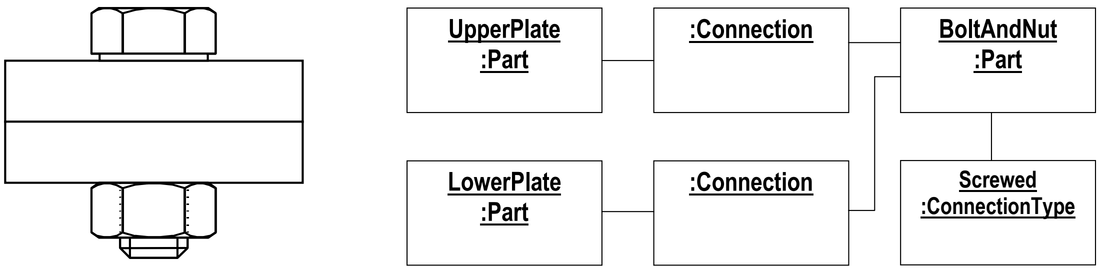


Figure 3. Simplified UML object diagram of two plates screwed together

The example in Figure 2 is a bearing force fitted to a shaft. The unnamed connection references the *ConnectionType*-object "ForceFitted". The example in Figure 3 shows how two plates screwed together would be modelled. Note that the *Part*-object "BoltAndNut" references the *ConnectionType*-object "Screwed", indicating its role as a fastener and that only the connections between the fastener and the parts that are connected by it are modelled (the fact that the two plates are in contact to each other is irrelevant in this case).
Since *Assembly*-objects neither reference *Connection*-, nor *ConnectionType*-objects, they can neither act as fastener, nor be directly connected to each other (although it is possible to model two connected assemblies by connecting the corresponding parts belonging to each assembly). Therefore, the bolt and the nut has to be modelled as a Part-object. This limitation, however, was accepted for the sake of the simplicity of the model.

2.4 Implementation

The implementation of the partial product model benefited of the fact that an object oriented model can be mapped quite easily to a relational model, which is the foundation of any relational database [Ambler, 2003]. Basically, each table represents a class, and each row of a table an instance of a class. Accordingly, the table columns correspond to the class properties. In the figure below, each box symbolizes a database table, containing a list of the column headers.

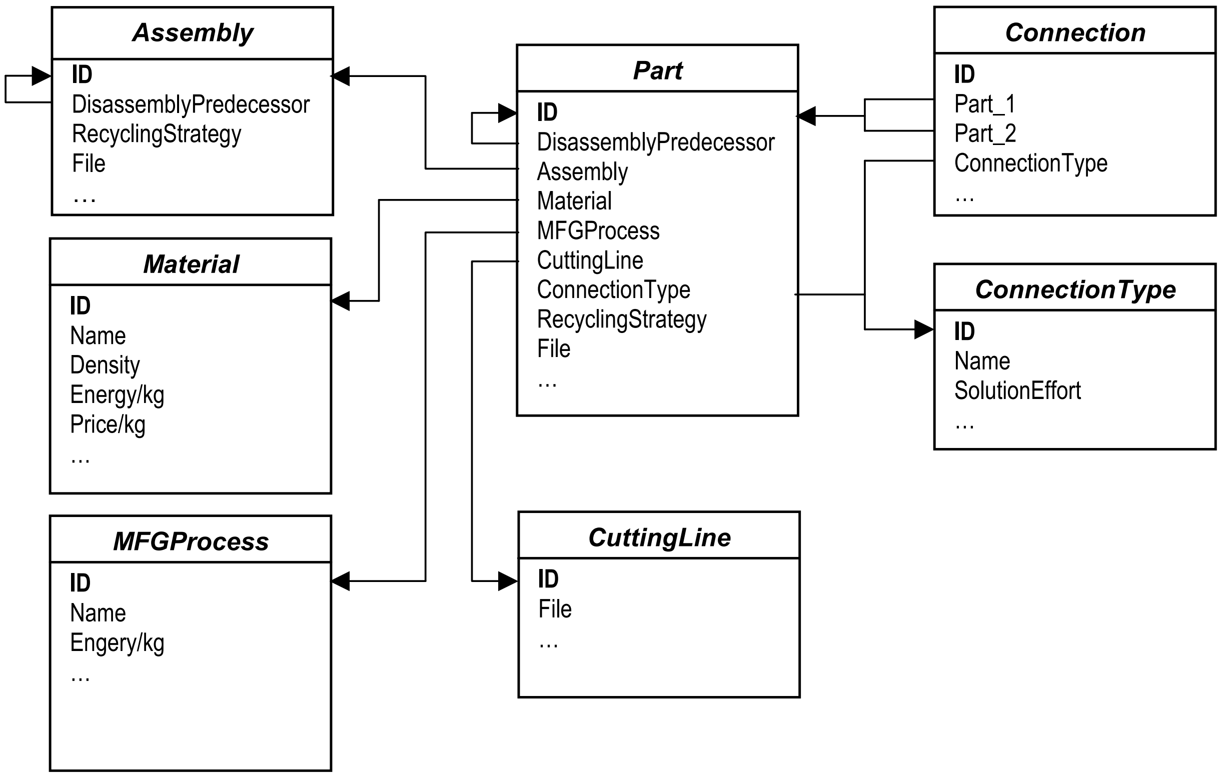


Figure 4. The product model as relational data structure (each box represents a database table)

In addition to the class properties of the object oriented product model, each table of the relational data structure contains the column *ID*, the "primary key" (marked bold in Figure 4). It serves as an unique identifier of each data set. Associations between objects (as described above) are implemented by primary keys being referenced by "foreign keys".

E.g. to indicate that a part belongs to a specific assembly, the field value at the column *Assembly* (the foreign key) in the row representing that part in the table *Part* would contain the value at the column *ID* (the primary key) in the row representing that assembly in the table *Assembly*. The database system ensures that no two data sets with the same primary key exist within the same table. It also monitors referential integrity, i.e. foreign keys only pointing to data sets that actually exist.

The attribute *File*, that appears in the tables *Assembly*, *Part* and *CuttingLine*, reflects an important implementation aspect of the partial product model: All geometry data is stored in the files of the CAD application. Therefore the implementation of the product model is independent of the CAD software used in the system.

3. System architecture

In order to provide a platform independent and network-compatible access to the product data – as described by the partial product model – a system architecture has been developed that is based on a CORBA client/server environment. The "Common Object Request Broker Architecture" is an open standard for developing distributed object oriented software applications [OMG, 2002]. In a CORBA environment, server applications implement objects that are accessed by remote client applications via a standardised interface. For the client applications, these remote objects behave largely like local objects.

Figure Figure 5 shows the architecture of the Design Support System. Each of the displayed components (Design Support Server, Interbase, SmartAgent, etc.) can run on individual computers in a network as well as all components on a single machine.

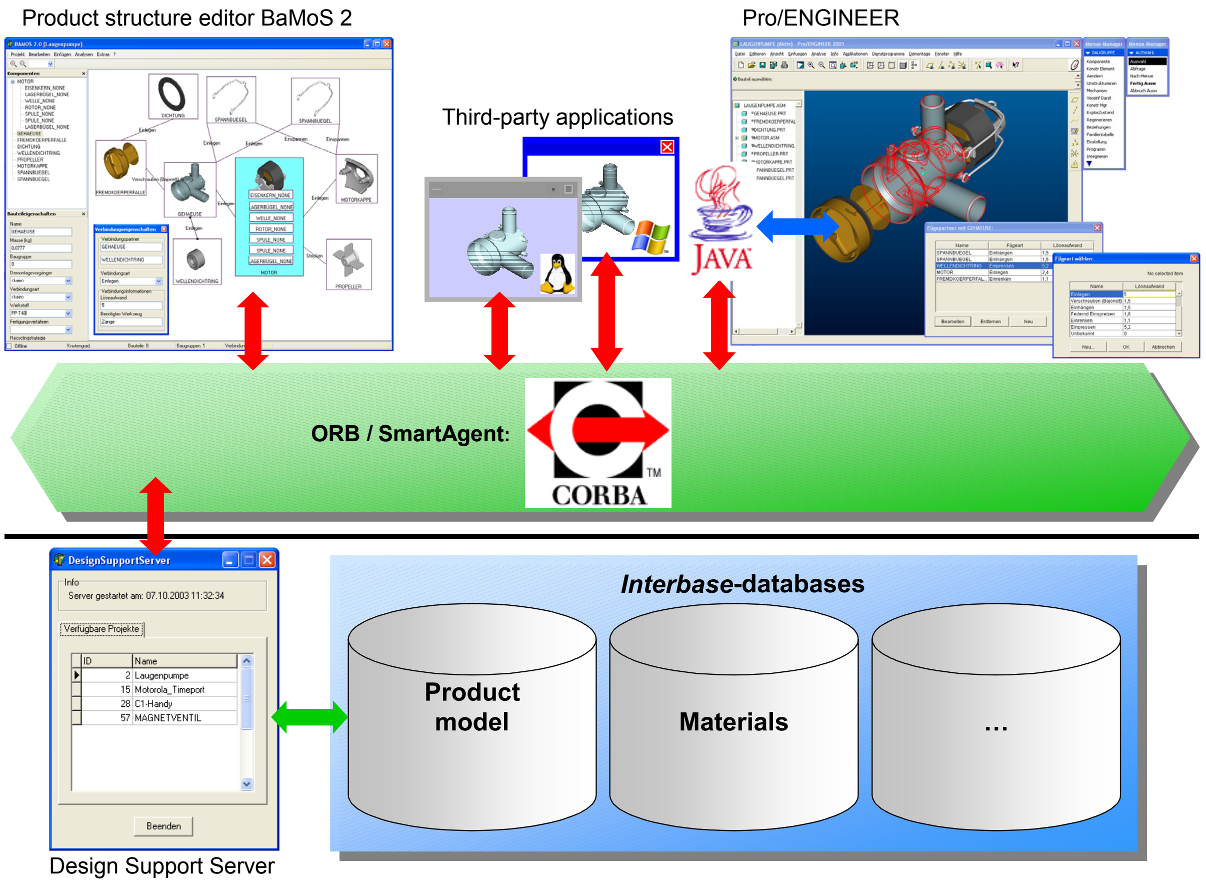


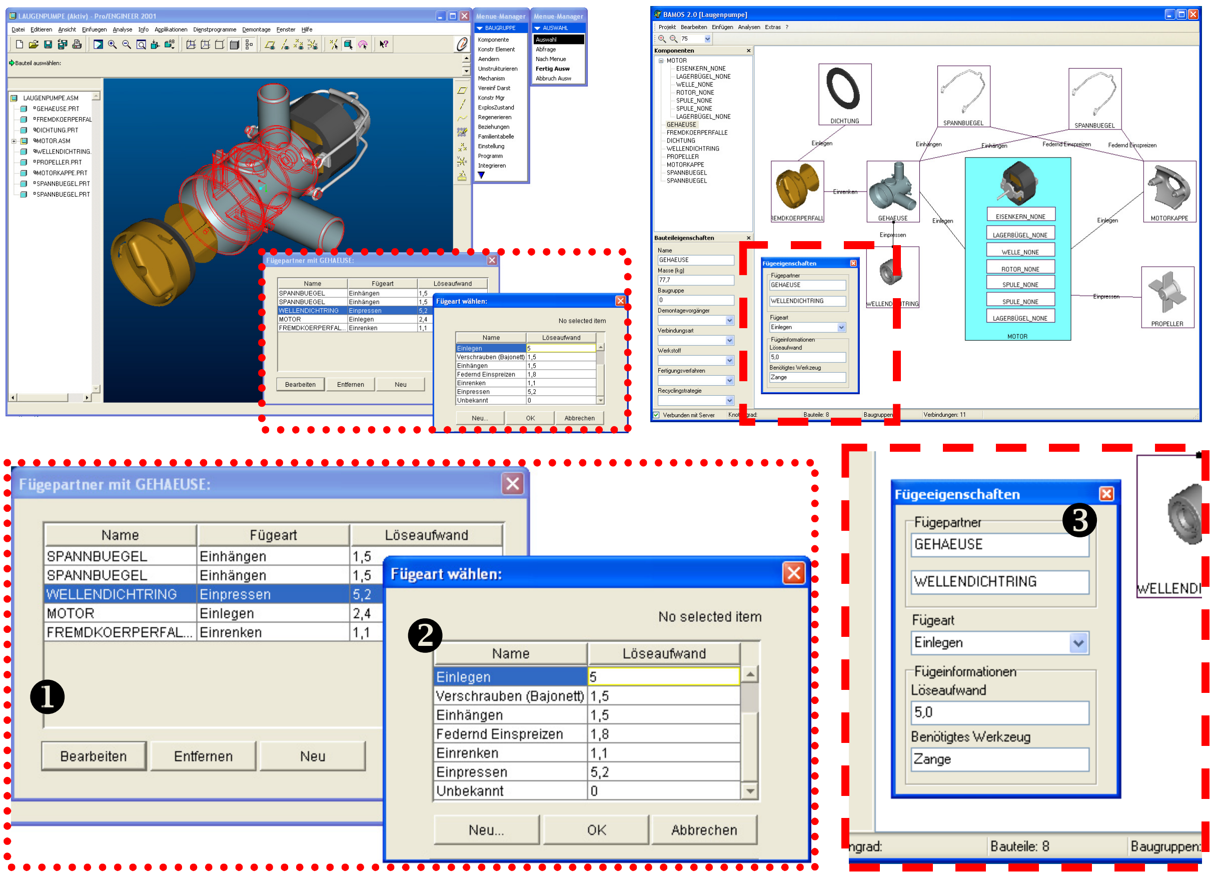
Figure 5. Architecture of the Design Support System

Centre of the system is the *Design Support Server*. Its main task is to map the data stored in the tables of the *Interbase* database system to corresponding CORBA-objects and to handle all client requests. These client requests, automatically invoked by the manipulation of a CORBA object, are interpreted by the server which, being the only application with direct access to the product data, returns the desired result (e.g. the mass or material of a specific part) or executes the according operation (e.g. connecting two parts). Therefore, the client applications of the Design Support System do not directly manipulate any product data, but invisibly communicate with the Design Support Server.
This kind of client-server communication is enabled by “Object Request Brokers” (ORB). These helper applications need to be active on each client platform and act as the direct interface with the remote objects. Since ORB software is available for almost any system environment (Windows, UNIX, etc.), third-party applications of any platform can be integrated.
The “SmartAgent” has a relaying function. Once contacted by an ORB, it searches the network for the server that actually implements the requested object. Unlike the ORBs, there has to be only one active SmartAgent within the network.
Adapting a program for integration into the Design Support System is achieved by using IDL files. The “Interface Definition Language”, which is defined by the CORBA specifications, describes the interfaces of the objects provided by the server, independently from the programming language of the client application. Since most development environments (e.g. Delphi or JBuilder) are capable of processing those IDL files, developers are relieved of the task of re-defining the object interfaces.

4. Example: interaction of Pro/ENGINEER with BaMoS 2

BaMoS 2 is an modelling/analysis tool for product structures (Baustrukturanalyse- und Modellierungs-System) based on [Radtke, 2000]. Its development also served as a reference example for writing software compatible with the Design Support System using IDL files. BaMoS delivers a graphical representation of the product's structure, basically displaying parts and assemblies as nodes, and connections as vertices. A number of analysis functions give information about the product's material composition, used connection types, disassembly effort, etc.

Using *J-Link*, PTC's Java-based programming interface, the user interface of Pro/ENGINEER was extended with functions related to disassembly and recycling. Users can now interactively define connections between parts, specifying connection type and solution effort, assign materials from the design support system's materials database, etc. By mapping the application's internal product model to the product model of the Design Support System (as hinted in Figure 5), changes made within Pro/ENGINEER, like defining a disassembly relevant connection, become instantly visible in other applications (e.g. BaMoS) and vice versa.



Fügepartner mit GEHAEUSE:

Name	Fügeart	Löseaufwand
SPANNBUEGEL	Einhängen	1,5
SPANNBUEGEL	Einhängen	1,5
WELLENDICHTRING	Einpressen	5,2
MOTOR	Einlegen	2,4
FREMDKOERPERFAL...	Einrenken	1,1

Fügeart wählen:

No selected item

Name	Löseaufwand
Einlegen	5
Verschrauben (Bajonett)	1,5
Einhängen	1,5
Federnd Einspreizen	1,8
Einrenken	1,1
Einpressen	5,2
Unbekannt	0

Fügeeigenschaften

Fügepartner: GEHAEUSE, WELLENDICHTRING

Fügeart: Einlegen

Fügeinformationen

Löseaufwand: 5,0

Benötigtes Werkzeug: Zange

Figure 6. Real-time interaction between Pro/ENGINEER (top left) and BaMoS 2 (top right)

Figure 6 demonstrates a possible interaction of the two applications, using a washing machine pump as a product example. Dialogue 1, which is invoked from within Pro/ENGINEER, lists all parts connected to the selected part ("Fügepartner mit GEHAEUSE" = Connection Partners with GEHAEUSE [case]) together with connection type ("Fügeart") and a disconnecting effort ("Löseaufwand") which is based on [Bruchhold, 1988]. Each list entry in this dialogue corresponds to a vertex connected to the same part as displayed by BaMoS. Users can connect a new ("Neu") part to the case or either remove ("Entfernen") or edit ("Bearbeiten") the connection to the part highlighted in the list. In the example shown, the user has chosen to edit the connection with the shaft seal

("WELLENDICHTRING"), changing the connection type from force fit ("Einpressen") to inlay ("Einlegen") (dialogue 2).
Once all changes are confirmed, the product data is automatically updated using the mechanisms described in the previous section. A user of BaMoS could now access the properties of the connection between the shaft seal and the case and see the updated information (dialogue 3).

5. Conclusion and outlook

In this paper, the current state of a software based system to support the design for disassembly has been presented. This Design Support System is based on a common partial product model that is capable of representing a large spectrum of information relevant to disassembly and recycling, which is used to describe centrally stored product data. To allow a wide range of tools and applications to access this data, a distributed object oriented approach has been chosen, based on CORBA as an open and accepted standard. Using Pro/ENGINEER and BaMoS 2 as an example, it was shown that within such a framework an existing standard application can be adapted to interact with custom-made software.

A lot of disassembly and recycling related software tools exist today and even more will probably become available in the future. Still, their usefulness as a whole is often limited by lacking possibilities to interchange product data – especially with existing applications, such as CAD or LCA software. This is due to two main causes: incompatible data and incompatible interfaces. Consequently, the designer often has to manually enter the output of one application as input for another. The software system that has been presented in this paper, intends to support the designer of disassembly and recycling-oriented products by relieving him of this task.

As to the incompatible interfaces, a technical solution has been found that, compared with similar technologies (e.g. the "Distributed Component Object Model" (DCOM) by Microsoft or the "Java/Remote Method Invocation" (Java/RMI) by Sun), harmonizes with the widest range of system environments and development platforms. The problem of incompatible data required to develop a disassembly- and recycling-oriented partial product model. Choosing an object oriented approach ensured a straightforward implementation of the software and compatibility with the interface solution (CORBA). Although the partial product model may not be complete, it is capable of representing important disassembly and recycling relevant product aspects, particularly the product structure. The example of Pro/ENGINEER shows that it is possible to integrate existing standard applications into the design support system, provided that they feature an object oriented programming interface. Still, this circumstance shall not mislead from the fact, that mapping one product model to another is obviously not the perfect solution. Not forcing designers to change tools that they are familiar with, on the other hand, should contribute to the acceptance of any design-related software environment.

On a more general level, the research presented in this paper is believed to give possible answers to the questions "What should product data look like?" and "How can different programmes access this data?". It would therefore be imaginable to use the technology of the Design Support System in other fields of computer aided design where these questions matter. Apart from further refining the product model and exemplarily integrating additional existing software (first of all, an LCA application), future work will concentrate on the evaluation of the system. Even though the technical solution is more or less complete, it is needless to say that its impact on the design process as well as on the development process of third-party applications still has to be explored.

Acknowledgement

This paper is based on research by the project “Support for the Design for Disassembly” that is part of the Collaborative Research Center 281 funded by the Deutsche Forschungsgemeinschaft (DFG). Its support is highly acknowledged.

References

Ambler, S., “The Elements of UML Style”, Cambridge University Press, Cambridge, 2002.

Ambler, S., “Agile Database Techniques : Effective Strategies for the Agile Software Developer”, John Wiley & Sons, Hoboken, NJ, 2003.

Bruchhold, I., “Untersuchung der Tragfähigkeit und des Füge- und Trennverhaltens von lösbaren Verbindungen”, Schriftenreihe Konstruktionstechnik, Vol. 15, W. Beitz (editor), Berlin, 1988.

Collaborative Research Center (CRC) 281: Disassembly Factories for the Recovery of Resources from Product- and Material Cycles, Partial Project D2, “Arbeits- und Ergebnisbericht 2001 – 2003“, Technical Universtiy Berlin, 2003.

Dose, J., Gries, B., Fleischer, G., Blessing, L., “Entwicklung nachhaltiger Produkte unter ökobilianziellen Gesichtspunkten”, Proceedings Design for X, H. Meerkamm (editor), Lehrstuhl für Konstruktionstechnik, Universität Erlangen, 2003, pp 63-70.

Grabowski, H., Anderl, R.., Polly, A., “Integriertes Produktmodell”, Beuth Verlag, Berlin, 1993.

Object Management Group (OMG), Common Object Broker Architecture (CORBA/IIOP), specification 3.0.2, OMG Inc., Needham, MA, 2002.

Pahl G., Beitz W., “Engineering Design – A Systematic Approach”, Springer-Verlag, London, 1996.

Radtke, M., “Bamos-Baustruktureditor”, DIN-Loseblattwerk “Umweltgerechte Produktentwicklung“, Beuth-Verlag, Berlin, 2000.

Dipl.-Ing. Bruno Gries
Technical University Berlin, Engineering Design and Methodology
Strasse des 17. Juni 135, 10623 Berlin, Germany
Telephone: +49-(0)30-314-23161, Telefax: +49-(0)30-314-26481
E-mail: gries@ktem.tu-berlin.de