Management of Product Data Using an Extensible Object-Oriented Query Language

Kjell Orsborn

Linköping University
Department of Computer and Information Science
Engineering Databases and Systems Laboratory
S-581 83 Linköping, Sweden

(Presented at the Sixth International Conference on Data and Knowledge Systems for Manufacturing and Engineering, DKSME '96)

Abstract. The concepts of product models and data management systems for engineering data are becoming increasingly important in industry to support rational product development and manufacturing. Considerable reductions in production costs and development time are possible, and efficient information management is becoming a strategic issue. This work shows how a state-of-the-art object-oriented and extensible (also referred to as object-relational) query language, AMOSQL, can be used for product data management. AMOSQL is part of a main-memory object-relational database system, AMOS, which combines high-level modelling with high execution efficiency. The AMOS architecture incorporates database functionality and conforms to the mediator approach forming an intermediate level between applications and data sources. By relying on general DBMS technology for product data management, it is possible to take advantage of built in database facilities for client-server communication, data sharing and distribution, concurrency control, etc.

An example presents an outline of a product model for single sheet-metal parts, how the query language can be used to model, retrieve and update the product model, and further how the query language can be extended with additional domain functionality. *Declarative models* are more compact and (de)composable than *procedural models* making the system more transparent and flexible. An application and data-independent representation further facilitate reuse and evolution of the design. Queries are automatically optimized by the DBMS and the query language also support advanced *ad hoc queries* concerning the contents of the database.

1 Introduction

This report treats product modelling using a state-of-the-art extensible and object-oriented (OO), also termed object-relational (OR) (Stonebraker and

Moore 1996), query language. The query language, AMOSQL, is part of the OR database management system (DBMS) AMOS, (Fahl et al. 1993). The idea is to show how next generation database techniques can be used in product data management. By using a DBMS as an intermediate level between applications and data sources, built in database facilities can be used to control data sharing and distribution, concurrency, etc. An example of a product data model for sheet-metal parts is modelled and implemented (Orsborn 1993). This model describes a subclass of the product models for mechanical designs that for instance are valid within the aircraft industry. The work has been carried out in cooperation with Saab Military Aircraft AB.

The concepts of product models and data management systems for engineering data are becoming increasingly important in industry and has also been paid a lot of attention in the research community as exemplified in (Meerkamm 1995) and (Mäntylä 1995). The goal is a rational product development and manufacturing and efficient information management is becoming a strategic issue. It is for instance possible to make considerable reductions in production costs and development time (Johansson and Wikström 1992).

A product model should be able to represent a sufficient level of structure and behaviour of the real product to make fair judgements of its characteristics in the real world and to be able to provide the product with these characteristics. In addition, it might be necessary to use several different types of product models for different parts or phases in the production process. To provide a general base for product data representation an international standard, the STEP standard (ISO 1992), is emerging. This standard is primarily a specification standard for data exchange and do not currently cover the implementation of product models. Object database technology can provide key technology for this task.

OO techniques are well suited to reduce system complexity and their evolution into OO DBMS, have further extended their applicability. Their abilities of representing and managing complex data and relationships make them especially suitable to engineering applications (Cattell 1991). The next generation of OO database technology, also referred to as OR database technology, integrates OO and relational database technology. It combines OO modelling capabilities with query language facilities.

Hence, OR presumes the existence of a relationally complete OO query language. In addition, it is expected that the DBMS can treat extensibility at both the query and storage management level. The AMOS OR DBMS is further a main-memory DBMS that combines high-level modelling with high execution efficiency. Declarative OO queries, as provided by AMOSQL, offers several advantages over conventional programming. A declarative description of product data is more transparent and flexible than a procedural and thus can increase the efficiency of modelling and managing domain models and their corresponding applications. Declarative modelling provides a compact and (de)composable description supporting a natural mapping of the problem domain. Query optimization techniques provided by the DBMS leaves execution strategy considerations to the system. Ad hoc queries can be used to access not predefined selections of product information. Specific modelling requirements from the problem domain can efficiently be taken care of by an extensible query language.

2 Product Models for Mechanical Engineering Design

2.1 Basic Description of the Product Model Concept

A product model is a conceptual model of a physical or abstract product, preferably implemented in a computing system. It provides an appropriate description of the characteristics and functionality of the product. For mechanical engineering products this includes a description of structure, form, content and behaviour. This description should be kept during the complete or parts of the life-cycle of the product, depending on the nature of the specific product.

The field of mechanical engineering contains a very broad spectrum of products, ranging from nuts and bolts to cars and aircraft. These products should be specified, designed, manufactured, maintained and possibly recycled. Besides technical issues there are other functions, such as marketing, selling etc., involved in the product process. During this process, i.e. the complete life-cycle of the product, the product model can be used as an information source to support different activities. The information needed for different purposes and in different phases varies a great deal and the product model is continuously evolving during this process.

2.2 Structure and Complexity of Product Models

Apparently a complete product model relatively soon becomes quite complex and includes a large amount of data. To reduce complexity of the description, it is necessary to divide the description into several subdescriptions which supports certain technology domains, life-cycle phases and representation levels of the product. Within mechanical engineering, there is production planning that needs manufacturing data whereas strength analysis needs data for loading and material conditions. However, there might also be shared data, such as geometrical data. Likewise, different product models might be required for different stages in the product process. Furthermore, a complete product can consist of a single part or be configured by many parts into subsystems that forms the whole. Even at the part level, several different product models might be required for different types of products. For example, (Johansson and Wikström 1992), reports on about 25 part types at Saab[1], such as sheet-metal parts, machined parts, composite parts, etc.

If we restrict the discussion to product models for single parts, there are still several aspects that have to be modelled. It might be convenient to refer to different substructures of the part by specific concepts (also known as *features*), such as holes, flanges, segments, etc. Further, the part has properties and relations to other parts that need to be modelled. There are also more abstract relations such as presentation and analysis models for a specific part.

2.3 Using Query Languages for Managing Product Models

To provide access to the information of the product model there must be some kind of access system. It is possible to use both programmed procedures and high-level query languages for this purpose. A query language is used to define, manipulate and retrieve information in a database. For instance, for retrieving some specific property of a product, you need not write dedicated program procedures for retrieving the expected information. Instead, a query can be formed in a high-level and data-independent query language which returns the information that fulfils the conditions of the query. Combining general query-language constructs with domain-re-

1. Saab is an abbreviation for SAAB-SCANIA AB, Saab Military Aircraft, Linköping, Sweden.

lated concepts provides a more problem-oriented communication. This approach to data management is more efficient compared to the use of conventional programming languages (Takizawa 1983). The combination of programming and query languages and their pros and cons for data management are further discussed in (Cattell 1991).

The technique of using OR query languages, like AMOSQL, is not only useful for usage of product modelling systems, but also the development, maintenance and evolution phases will gain in efficiency. Declarative OR queries offers several advantages over conventional programming for product data management.

- *Declarative models* become easier to describe, inspect and understand and thus become more transparent. Declarative modelling with an OO query-language is compact and (de)composable and makes the domain modelling very flexible and powerful.
- The query processor in a DBMS provides automatic optimization of queries. This leaves the problem of finding an efficient execution plan to the system.
- Query languages also make it possible to make advanced *ad hoc queries* concerning the contents of the database. This might be demanded by advanced users and is quite useful since it is impossible to foresee the complete information need.
- Advanced OO query-languages also provide *object views* capabilities. In AMOSQL, views are supported through derived functions, where a function is uniformly invoked independent of whether it represents stored or derived data. This makes it possible to change the underlying physical object representation without altering the access queries. Thus, data independence and evolution are supported.
- *Extensible* query languages provides powerful means for flexible domain modelling. This include support for user defined types and operators. Further issues in extensibility include the ability to define new data structures and operators, and the ability of the query processor to handle optimizations of these new constructs.
- The query language can also support active behaviour that can be used for advanced consistency checking of product models. An active rule system can monitor and react to predefined database events. Integration

of active rules in the AMOS DBMS is treated in (Risch and Sköld 1992).

The advantages of these features are, of course, varying for different phases in the system or application life-cycle and for different types of application users and system developers. Declarative modelling, using an OR query language, will however support an incremental and iterative development, maintenance and evolution of product models as well as product modelling systems. It will also facilitate a reuse and evolution of design in its ability of application and data independent representation.

2.4 Standards for Product Modelling

An ongoing work is specifying an international standard for management of product information. The International Standard ISO 10303 "Industrial Automation Systems and Integration - Product Data Representation and Exchange" (ISO 1992) that is usually referred to as STEP, has the objective to

> *"provide a neutral mechanism capable of describing product data throughout the life cycle of a product, independent from any particular system".*

STEP should provide a uniform language for management of product information independent of the discipline involved and where different levels of usage could be ranging from the application specific to the inter-enterprise level. ISO 10303 is organized in a series of parts, where the series involves: *overview, description methods, implementation methods, conformance testing methodology and framework, integrated generic resources, integrated application resources, application protocols,* and *abstract test suites.* The overview series describes the structure and contents of the standard and the description methods series includes the description of the formal data description language EXPRESS, on which the STEP standard is based, and also graphical representation notations, e.g. by means of EXPRESS-G. Further, the implementation methods series intend to specify different implementation techniques for realizing the data sharing, including physical file exchange, application programming interfaces - the STEP data access interface (SDAI), and database implementations. The conformance testing methodology and framework together with the abstract test suites should define the requirements and testing procedures to apply to an implementation to judge if it is in conformance with the ISO 10 303 application protocol. The integrated generic resources, integrated application resources, and application protocols include the actual conceptual schema’s for product information. These ingredients make it possible to specify, represent, and exchange product information in a formalized and standardized way, independent of the system, software or discipline involved. However, two different systems must be able to represent the same types of product data that they would like to exchange.

The idea of a standard for representing and communicating product information is of great importance for the development and efficiency of enterprise and engineering information management. It simplifies the management and communication of product information in an inter-enterprise or -organization situation as well as within enterprises. A standard will also make industry less dependant of certain software or hardware vendors and thus stimulate further evolution of product management systems. The STEP standard was originally designed to provide a uniform and file-based exchange format for product data. However, there is ongoing work that addresses data exchange based on database technology that might result in query language-based part of the SDAI data access interface. A query language can simplify the representation of application protocols that might be defined as views of the product model. There is also important to complement the specification and access standardisation with research on how to implement product models. This initial work is intended to show how general database techniques can support this task. There is also ongoing work that studies the representation of STEP schemas within the AMOS DBMS.

3 The AMOS DBMS and the AMOSQL Query Language

AMOS (Fahl et al. 1993), is a research prototype of an OR DBMS and a descendant of the WS-IRiS DBMS (Litwin and Risch 1992). WS-IRiS is further a derivative of Iris (Fishman et al. 1989). AMOS is a main-memory DBMS, i.e. it assumes that the entire database is contained in main-memory. It includes the AMOSQL OR (i.e. OO and extensible) query language that is used to model and interface the database. AMOSQL is a derivative of OSQL (Lyngbaek et al. 1991). Furthermore, AMOSQL is a functional language, originating from DAPLEX (Shipman 1981).

AMOS provides DBMS facilities including a local database, a data dictionary, a query processor, transaction processing, and remote access to data sources. It conforms to the mediator approach (Risch and Wiederhold 1991) that introduces an intermediate level of software between databases and their use in applications and by users. The purpose of a mediator is to query, monitor, transform, combine, and locate desired information between applications and data sources. A data source can be data stored in conventional DBMSs, other mediators, as well as data obtained by executing some program. Since mediators "understand" application terminology as well as database terminology they can combine, and take advantage of, both high-level domain-specific data interaction and efficient data management. The AMOS architecture permits that the DBMS is embedded within applications or configured as a client-server system. Hence, applications can take advantage of the DBMS facilities for storing, retrieving, or exchanging data. There is currently ongoing work that uses AMOS as an embedded database (Orsborn 1994), for integration of heterogeneous data sources (Fahl 1994), and as a distributed DBMS (Werner 1996).

The data model of AMOS consists of the three basic constructs: *objects, types,* and *functions.* Concepts in an application domain are represented as objects. There are two types of objects in AMOS. *Literal objects*, such as boolean, character string, integer, real, etc., is self identifying. The other type, called *surrogate objects,* has unique object identifiers. Surrogate objects represent physical, abstract, external, or internal concepts. Examples of surrogate objects might be mechanical components and assemblies such as skin panel or wing in an aircraft design, engineering elements such as flange and web, geometrical elements, etc. Also system-specific objects, e.g. types and functions, are treated as surrogate objects.

Types are used to structure objects according to their functional characteristics and types are in themselves related in a type hierarchy of subtypes and supertypes. Subtypes inherit functions from supertypes and both subtypes and objects can have multiple supertypes.

Functions are defined on types, and are used to represent attributes of, relations among, and operations on objects. Examples of functions for these different categories might be length, distance, and rotate_object. It is possible to define functions as stored, derived, procedure or foreign. A stored function has its extension explicitly stored in its data structure, whereas a derived, procedure, or a foreign function has its extension defined in an AMOSQL query, a general AMOSQL procedure, or a foreign program procedure, respectively. Functions can be overloaded on different subtypes (i.e. having different implementation for different types), defined as one- or many-valued, and are automatically supporting invertibility and late binding (Flodin 1995). Stored functions can be explicitly updated using update semantics, but other functions need special treatment for update. The AMOSQL language supports aggregate and recursive functions and can be seamlessly extended with new foreign functions by calling external programming languages like C or LISP.

AMOSQL provides constructs for typical database tasks, including data definition, population, updates, querying, flow control, and session and transaction control. Data schemas can be defined, modified, and deleted by means of AMOSQL statements both statically and dynamically. The following example creates a type engineering_element, a function name, an instance of engineering_element named "eng-el_1". The first select statement then queries the database for all engineering_elements, the second extracts a specific engineering_element, and the third extracts the name of a specific object.

```
create type engineering_element;

create function name(engineering_element el) ->
                 charstring as stored;

create engineering_element(name) instances
          :e1 ("eng-el_1");

select e for each engineering_element e;

select e for each engineering_element e
                   where name(e) = "eng-el_1";

select name(:e1);
```

A more thorough presentation of data management capabilities in AMOS and AMOSQL is presented in (Karlsson et al.1994).

4 A Product Model for Sheet-Metal Parts

This section presents a conceptual model of a simple product model for sheet-metal parts and its implementation in AMOS. It is shown how product information could be accessed by using an OR query-language, AMOSQL. Typical product information modelled and used in the examples is extracted in cooperation with Saab mainly from (Johansson and Wikström 1992) and (Wall 1990), with some completions and adaptions to suit this representation. It is not our intention to provide a complete product model for sheet-metal parts. It has rather been an aim to include the concepts and interrelations necessary to describe the logical structure of a typical part and to show how this information can be modelled and accessed by the query-language.

4.1 A Conceptualization of Sheet-Metal Parts

In the Saab specification of product models for sheet-metal parts, (Johansson and Wikström 1992), a product model has been divided into *a technology model* and *a geometrical model.* The geometrical model is thought to include mainly CAD information describing a product's physical form. This geometry information was not intended to be stored in the product database but to be referenced through "pointers" in the database to geometrical CAD objects. Form might be viewed as a type of physical structure as opposed to logical structure covered by the technology model. This work mainly concerns the representation of the technology model and the representation of the physical geometry is therefore left out in the rest of this section. However, it would also be possible to integrate the geometrical information in the product database, if the hardware and software configuration could provide sufficient processing efficiency. In (Orsborn 1994) is shown that this constraint can be released by the emerging field of main-memory databases.

The modelling and implementation have been centred around a specific example presented in Figure 1. There was a basic set of concepts suggested by Saab including: part, web, flange, joggle, segment, face, bottom, edge, border, hole, etc. Along with these concepts there were typical relations and queries that were interesting to model in this context. The complete implementation is provided in (Orsborn 1993).

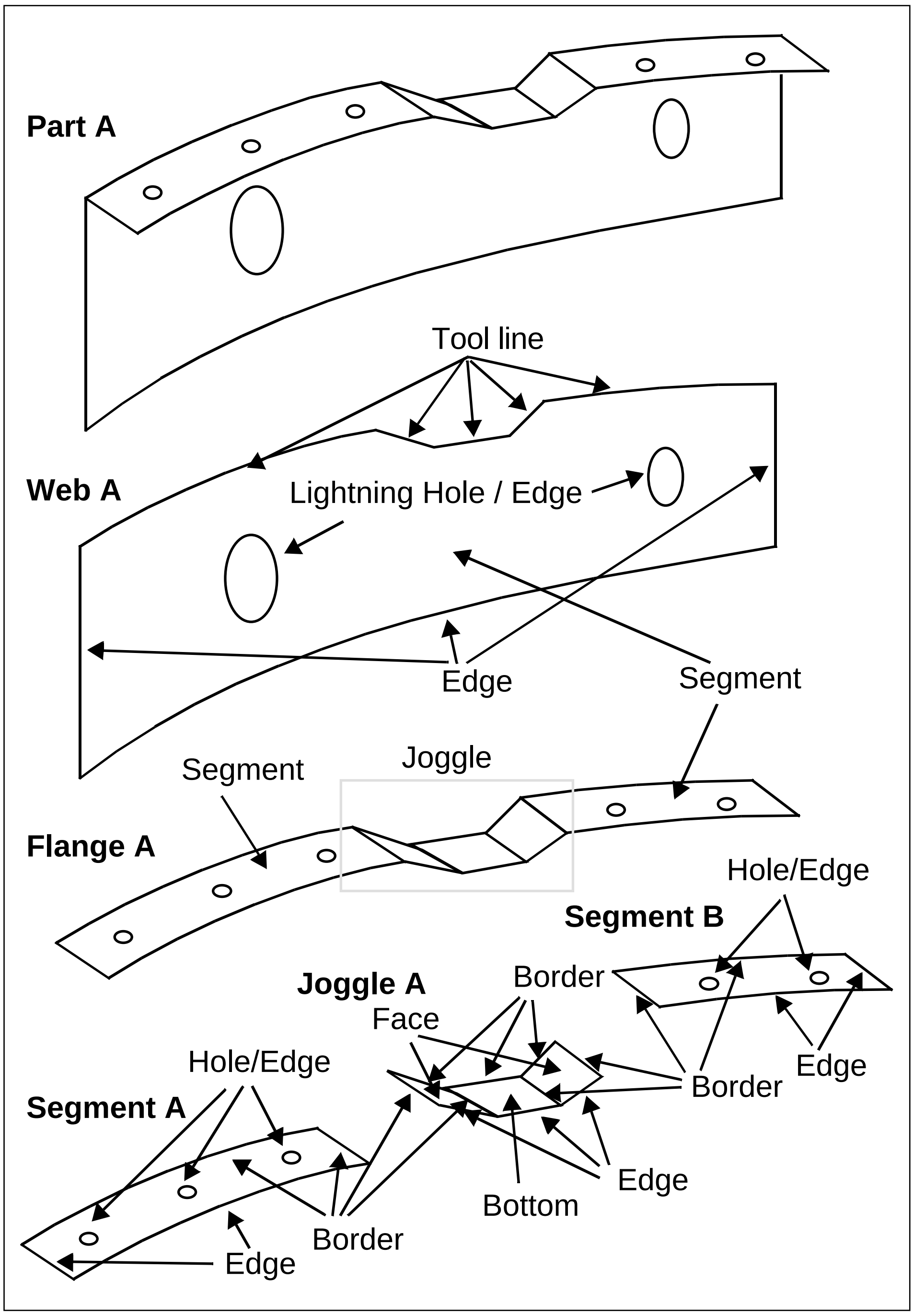

Figure 1. The conceptual structure of the modelled sheet-metal part.

Concept Structure and Modelling. A few definitions from (Johansson and Wikström 1992) need to be reproduced to facilitate the interpretation of this description. These include:

- *Engineering element* (EE): A concept that has an engineering meaning, a form, and a position. An EE has a technological and a geometrical part. Further, an EE may be composed of subordinate EE’s.
- *Relation* (R): Expresses a connection between EE’s.

A graphical view of the engineering concepts used for describing an instance of the sheet-metal part is presented in Figure 1. All concepts in this picture are classified as *engineering elements*. This set of concepts can be classified further into specialized subsets where each subset shares a common set of properties. This hierarchical specialization can be modelled by means of is-a relations as in Figure 2, where a subset is a specialization of a superset. In AMOSQL this relationship is implemented as a type structure with supertypes and subtypes, exemplified by:

```
create type composite_engineering_element subtype of
              engineering_element;

create type part subtype of
              composite_engineering_element;
```

The is-a relation has a semantic direction from subtype to supertype. Implemented as a type function in AMOS it works as an undirected relation and can be used in both directions.

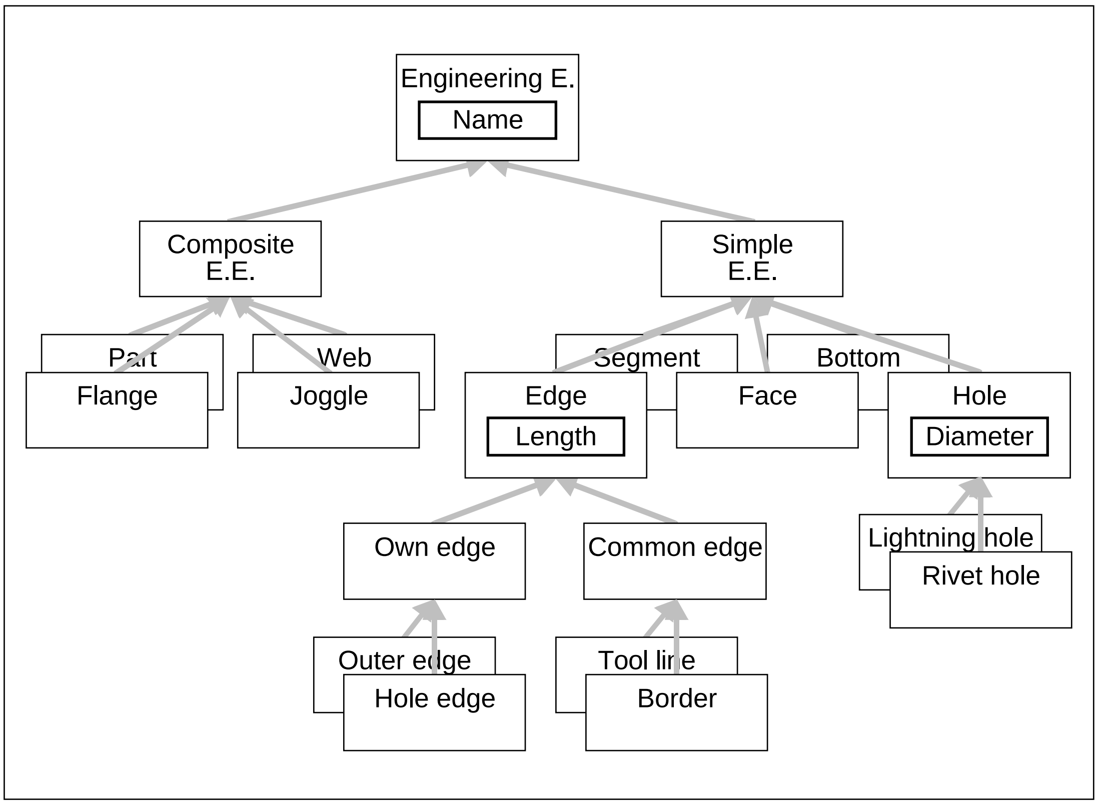

Figure 2. *Classification of concepts for sheet-metal parts. The grey arrows represent is-a relations.*

Properties, such as name, diameter, and length, of objects are defined as attributes on a certain class. These are implemented by means of stored functions in AMOSQL and can be exemplified by:

```
create function diameter(hole h key)-> real r as stored;
```

Updates are made by set, add, or remove statements, such as:

```
set diameter(:hole_1) = 20.0;
```

Domain Relations and Modelling. As for the engineering concepts in the previous section it might be convenient to define a few domain relations according to (Johansson and Wikström 1992). These are:

- *Consists of* (CO): This relation identifies those elements that together form the originating element.

- *Modifies* (MF): This relation is used when the creation of an element implies that an existing element will be changed. It identifies the added (modifying) element, the elements that are modified and the borders between the added and the modifying elements. A removal of the added element should imply a removal of the MF relation. Compare with the Replaces relation.
- *Borders upon* (BU): The BU relation is identical to the MF relation except that the elements concerned, are not modified.
- *Replaces* (RP): A RP relation is used in conjunction with the MF relation to connect the added element with the (part of the) original element it replaces.
- *Surrounded by* (SB): SB relations are used to connect an element to surrounding elements. For example, a flange segment is surrounded by edges and borders.
- *Lies in* (LI): This relation is used to associate an element to an existing element. The existing element is, strictly speaking, modified by the associated elements but it is usually not necessary to consider the change. Rivet holes and lightning holes are typical examples of such elements.

Relations are usually modelled by functions in AMOSQL. They are by default multi-valued and invertible. By introducing cardinality constraints, they can be restricted to single-valued or combined variants. The relations described above are presented in Figure 3. As for the is-a relation described earlier, these also include a semantic direction but they can be used in both directions in their implementation. A relation is expressed by a polyline with a triangle that shows the semantic direction with its top. The cardinality is expressed by a “fork” for a multi-valued relation and the plain line for single-valued.

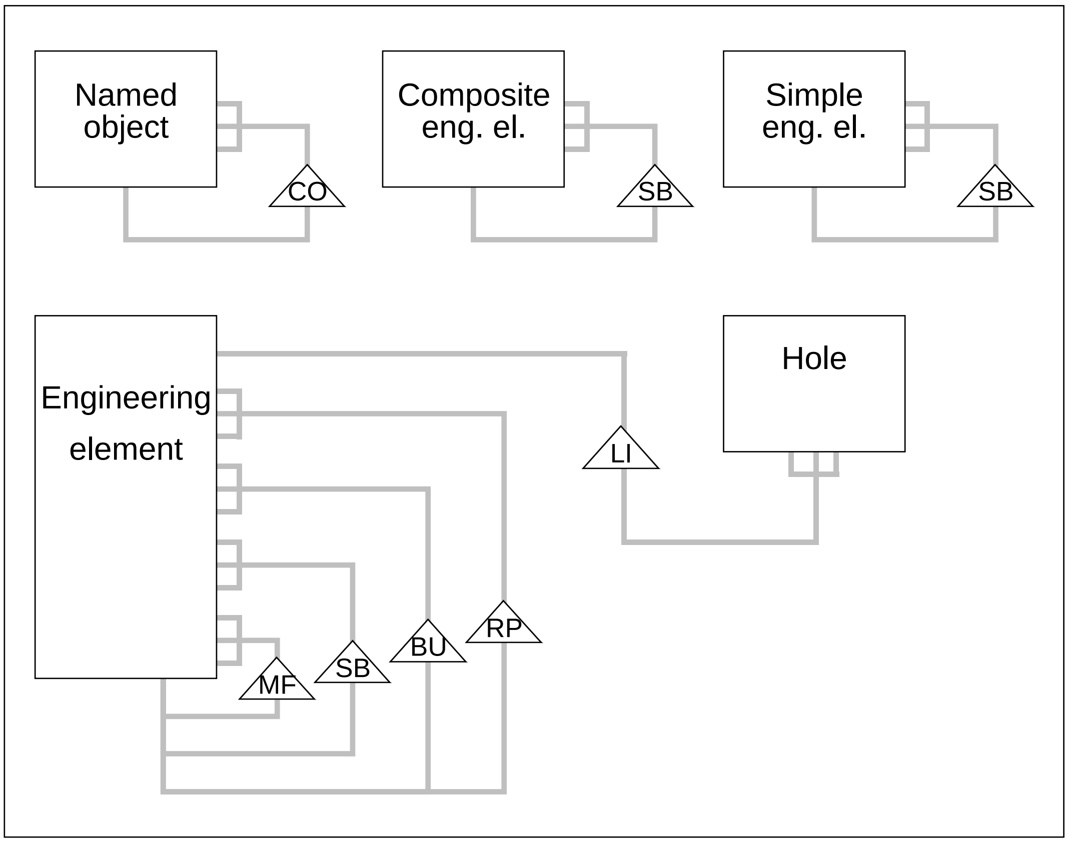

Figure 3. *Additional relations modelled for the sheet-metal part. These are consists-of (CO), surrounded-by (SB), modifies (MF), borders-upon (BU), replaces (RP), and lies-in (LI).*

The application scope of relations is automatically maintained by the system since functions are defined over types. Modelling of relations as functions in AMOSQL will, along with conceptual classification, reduce complexity of the resulting (product) model. In comparison to, for instance, a relational-oriented modelling technique this OO technique will result in a less complex, and a more flexible, model.

4.2 Querying the Product Model

There were several examples of typical queries for accessing a sheet-metal product model in (Johansson and Wikström 1992). Some examples of are listed below in plain english followed by the corresponding AMOSQL query and the query result. It should be noted that it is not necessary to use the name function to access the objects below. It is only used to simplify the interpretation of the examples.

1. **Which name has the engineering element that Joggle A consists of?** This query exemplifies how several functions (the name and consists_of functions) can be composed in queries. The second application of the name function illustrates how functions can be applied in the inverse direction.

```
select name(consists_of(j)) for each joggle j
       where name(j) = 'Joggle A';

<“Face A”> <“Face B”> <“Bottom A”>
```

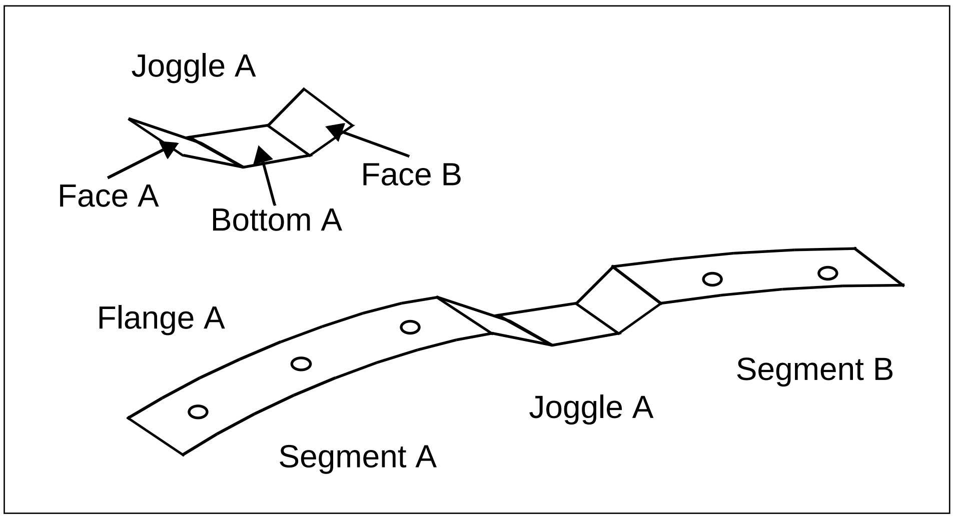

Figure 4. *Illustration for facilitating the interpretation of queries 1, 2, 3, 4, 5, and 7. Edge and border names are not provided since they are evident.*

2. **Which engineering element is modified by Joggle A?** Here, the where clause is combined by two conditions.

```
select name(e) for each engineering_element e,
                        border g, joggle j
     where name(j) = 'Joggle A' and modifies(j) = <e,g>;

<“Segment A”> <“Segment B”>
```

3. **Which engineering element modifies Segment A and Segment B?** Here, the where clause involves conditions on several objects.

```
select name(e) for each engineering_element e, border
         g1, border g2, segment s1, segment s2
       where name(s1) = 'Segment A' and
```

```
             name(s2) = 'Segment B' and
             modifies(e) = <s1,g1> and
             modifies(e) = <s2,g2>;
```

```
<“Joggle A”>
```

4. **Which is the border that modifies Segment A or Joggle A?** This where clause includes a disjunction.

```
select name(g) for each border g, segment s, joggle j
       where name(s) = 'Segment A' and
             name(j) = 'Joggle A' and
             modifies(j) = <s,g> or
             modifies(s) = <j,g>;
```

```
<“Segment A/Face A border”>
```

5. **How many rivet holes lie in Segment A?** This query exemplifies the use of the count aggregation operator and nested queries, where the result of a subquery is counted.

```
select count((select name(h) for each rivet_hole h,
                                      segment s
       where name(s) = 'Segment A' and lies_in(h) = s));
```

```
<3>
```

6. **What is the total length of the edges surrounding Segment A in Flange A?** Another aggregation that makes a sum calculation of the results of a subquery. Arithmetic operations can also be included within the where clause.

```
select sum((select length(k) for each edge k, segment s,
                                      flange f
         where name(f) = 'Flange A' and
               name(s) = 'Segment A' and
               consists_of(f) = s and
               surrounded_by(s) = k));
```

```
<1400.>
```

7. **Which edges surrounds Flange A?** See text below.

```
select name(surrounded_by(e)) for each
         engineering_element e
       where name(e) = 'Flange A';
```

```
<“Face edge A”>
<“Bottom_edge A”>
<“Face edge B”>
<“Segment_edge A1”>
<“Segment_edge A2”>
<“Segment_edge B1”>
<“Segment_edge B2”>
<“Bottom A/Segment C tool_line”>
<“Segment A/Segment C tool_line”>
<“Segment B/Segment C tool_line”>
<“Face A/Segment C tool_line”>
<“Face B/Segment C tool_line”>
```

8. **Which object id (OID) has the engineering element type that Segment_edge B1 belongs to?** This is an example of querying the database schema.

```
select parent(e) for each engineering_element e
       where name(e) = 'Segment_edge B1';

<OID[“OUTER_EDGE”:152] >
```

The seventh query applies the surrounded_by function on Flange A which is a composite engineering element. This means that the query must apply the surrounded_by function recursively on subsequent levels and filter out appropriate surrounding elements at each level. If the query would have been applied on a simple engineering element, the surrounded elements could have been extracted immediately. Hence, the function surrounded_by acts differently on simple and composite engineering elements respectively. This is implemented by overloading the surrounded_by on both types, which looks like:

```
create function surrounded_by(
                 simple_engineering_element ee1) ->
                 engineering_element ee2 as stored;

create function surrounded_by(
                 composite_engineering_element ee1) ->
                 engineering_element as
                select exclusive((select ee2 for each
                      engineering_element ee2,
                      engineering_element ee3
                    where consists_of(ee1) = ee3 and
                         surrounded_by(ee3) = ee2));
```

The definition of the surrounded_by relationship on the composite engineering element type include an exclusive operator that filters out elements appearing only once in a multiset. This operator is implemented by a foreign function as an extension to AMOSQL. It shows how conveniently and seamlessly the query language can be extended to meet domain requirements. Additional examples are provided, together with the complete implementation code in (Orsborn 1993).

5 Summary

The present work has shown how an OR query language, AMOSQL, can be used for management of product data. An example presents an outline of a product model for single sheet-metal parts. It include several examples on how the query language can be used to model, retrieve and update the product model. Query examples show specific facilities of the AMOSQL language, like function composition, invertibility, and overloaded late bound functions. It is further shown how the extensibility of the query language facilitates modelling of domain-oriented additional functionality, such as the incorporation of the exclusive operator. Ongoing work aims at representing STEP schemas in the AMOS DBMS.

The advantages of the product model concept for managing engineering data are reported on in (Johansson and Wikström 1992), and include higher efficiency (reduced costs, lead times, etc.) in the production process. Some subprocesses for generation of operation lists are expected to be reduced by as much as 70-80%. Besides these explicit advantages in *using* such a system for product modelling, it is important to point out the expected increase in efficiency for *development* and *maintenance* of product information. Additionally, these considerations are also valid for development and maintenance of the supporting product modelling system.

The use of an OR query language for product data management offers several advantages over conventional programming including declarative modelling, query optimization, ad hoc queries, data independence and evolution, and extensibility. This high-level and declarative modelling and manipulation of product data will also facilitate a reuse and evolution of design due to application and data independent representations. It is argued the this will increase the transparency, flexibility and overall efficiency of product modelling systems.

The AMOS architecture incorporates database functionality and conforms to the mediator approach that forms an intermediate level between applications and data sources. It is argued that by relying on general DBMS technology for product data management, it is possible to take advantage of built in database facilities like client-server communication, data sharing and distribution, concurrency control, etc.

Finally, it is the author’s opinion that OR query languages and DBMS technology, together with emerging standards for product modelling, will play an important role in future product data management.

6 References

Stonebraker, M., and Moore, D. 1996, *Object-relational DBMSs: the next great wave*, Morgan Kaufmann Publishers, Inc.

Cattell, R. G. G. 1991 (reprinted with corrections 1992), *Object data management: object-oriented and extended relational database systems*, Addison-Wesley Publishing Company, Inc.

Fahl, G., Risch, T. and Sköld, M. 1993, AMOS - an architecture for active mediators. Presented at the Workshop on Next Generation Information Technologies and Systems (NGITS’ 93), Haifa, Israel, June 28-30.

Fahl, G. 1994, Object views of relational data in multidatabase systems”, Licentiate Thesis LiU-Tek-Lic 1994:32, Linköping University, Linköping.

Fishman, D. H., Annevelink, J., Chow, E. Connors, T., Davis, J. W., Hasan, W. Hoch, C. G., Kent, W., Leichner, S., Lyngbaek, P., Mahbod, B., Neimat, M.A., Risch, T., Shan, M. C. and Wilkinson, W. K. 1989, Overview of the Iris DBMS. In Kim, W. and Lochovsky, F. H. (eds.): *Object-Oriented Concepts, Databases, and Applications* (ACM Press, Addison-Wesley).

Flodin, S. 1995, Processing object-oriented queries with invertible late bound functions, Proceedings of the 1995 Conference of Very Large Databases.

ISO 10303-1, 1992, Product data representation and exchange - part 1: overview and fundamental principles. ISO CD 10303-1, International Organization for Standardization.

Johansson, J. and Wikström, R. 1992, The future product development process for aircraft of the future at Saab Military Aircraft. Technical Report TUNJ 91:072E, SAAB-SCANIA AB, Linköping, Sweden.

Karlsson, J., Larsson, S., Risch, T., Sköld, M., Werner, M. 1994, AMOS users’s guide. CAELAB Memo 94-01, Linköping University.

Litwin, W. and Risch, T. 1992, Main memory oriented optimization of OO queries using typed Datalog with foreign predicates. *IEEE Transactions on Knowledge and Data Engineering*, **4**.

Lyngbaek, P., et al. 1991, OSQL: a language for object databases. Technical Report HPL-DTD-91-4, Hewlett-Packard Company.

Meerkamm, H. 1995, Product modelling: a prerequisite for effective product development. Proceedings of Product models -95, SIG-PM, ISBN 91-7871-541-5.

Mäntylä, M. 1995, Utilization of product models in product preparation. Proceedings of Product models -95, SIG-PM, ISBN 91-7871-541-5.

Orsborn, K. 1993, Modelling of Product Data Using an Extensible O-O Query Language. Technical Report LiTH-IDA-R-93-15, Department of Computer and Information Science, Linköping University, Linköping Sweden.

Orsborn, K. 1994, Applying Next Generation Object-Oriented DBMS to Finite Element Analysis. In Witold Litwin, Tore Risch (eds.): Application of Databases, 1st Int. Conf., ADB-94, Vadstena, Sweden, June 21-23, 1994 (Proceedings), Lecture Notes in Computer Science, Springer Verlag, ISBN 3-540-58183-9.

Risch, T. and Sköld, M. 1992, Active rules based on object-oriented queries. LiTH-IDA-R-92-35, Linköping University. Also in a special issue on Active Databases of *IEEE Data Engineering*, 1992.

Risch, T. and Wiederhold, G. 1991, Building adaptive applications using active mediators, Proceedings of Database and Expert Systems Applications (DEXA‘91).

Shipman, D. W. 1981, The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*, **6**, 140-173.

Takizawa, M. 1983, Distributed database system JDDBS. In *JARECT Computer Science & Technologies*, **7**, (OHMSHA & North Holland), 262-283.

Wall, H. 1990, Product modelling at Saab Aircraft, results and experiences. Technical Report TULB 90-23, SAAB-SCANIA AB, Linköping Sweden.

Werner, M. 1996, Multidatabase integration using polymorphic queries and views. Licentiate Thesis LiU-Tek-Lic 1996:11, Department of Computer and Information Science, Linköping University.