Presented at IDEAS'02 - International Database Engineering and Applications Symposium, Edmonton, Canada July 17-19, 2002

Completing CAD Data Queries for Visualization

Milena Gateva Koparanova and Tore Risch
Uppsala Database Laboratory
Department of Information Technology
Uppsala University
Sweden
{firstname.lastname}@it.uu.se

Abstract

A system has been developed permitting database queries over data extracted from a CAD system where the query result is returned back to the CAD for visualization and analysis. This has several challenges. First, CAD data representations use complex object-oriented schemas and the query language must be object-oriented too. Second, the query system resides outside the CAD system and must therefore use standardized data exchange formats for interoperability with the CAD. ISO STEP standard exchange formats are used for the exchange. Third, a CAD system cannot import an arbitrary object structure but places restrictions on the imported objects to be acceptable. Therefore, the query system must complement the query results in order to produce an acceptable CAD model, called the model completion of the query. These problems have been solved using an extensible object-relational query processor. The system also supports queries combining CAD data with data from other data sources.

1. Introduction

CAD systems such as I-DEAS [30] and Pro/ENGINEER [27] allow engineers to build design models that may contain very large amounts of data with complex object-oriented structures. During the engineering design process there is often need to analyze these models and select components and sub-models having certain properties. One way to support this is to regard the contents of such engineering models as databases that are queried using a database query language. A system offering techniques for query processing of engineering data extracted from CAD models has been developed, which is called the Engineering Mediator Query system (EMQ). EMQ allows data to be extracted from CAD models for subsequent analysis and querying.

Data in CAD systems are represented using complex Object-Oriented (OO) data representations. In order to minimize information loss and maximize query expressibility, the query system needs also to be OO. EMQ therefore uses an OO data model and query language based on an extended subset of SQL-99. The main enabling technology for querying engineering data is the kernel of the Amos II mediator database system [28]. EMQ uses the query language AmosQL of Amos II, with certain extensions.

The engineering queries can contain functions that match and select objects in the original CAD model. Different query types are possible, including queries on the hierarchical product structure and administrative information, as well as queries on the geometric and topological data defining the shape of a product. For the latter class of queries it is important to be able to return the query results back to a CAD system for visualization and further analysis. However, the CAD system can only visualize data that forms a complete geometric representation of the shape of some part. EMQ therefore has to extend the query result with certain objects that the CAD requires in order to make the former an acceptable CAD model. We call such an extended query result *model completion* of the query. An algorithm for the CAD data query completion has been developed and implemented in EMQ in order to accomplish the full cycle from CAD representation through a query system and back to the query result visualized in the original or other CAD system.

Such a system needs tools for data exchange with CAD systems. The ISO-10303 Standard for Product Data Representation and Exchange [14], known as STEP, provides standardized formats for data exchange with CAD and other engineering systems. The STEP standards use separate meta-data descriptions (schemas) of the

exchanged data formally expressed in the EXPRESS information modeling language [31]. For a given meta-data description the data can then be exchanged using the STEP/Part 21 standard format [17] for data exchange files. In database terminology, EXPRESS is a *data model*, while an EXPRESS *information model* contains meta-data descriptions of data about some portion of the real world and corresponds to a *schema* (possibly with subschemas).

Our system can read an information model in EXPRESS and translate it to a corresponding OO database schema. Given this OO schema, the EMQ system can then access standard data exchange files exported from the CAD using its standardized STEP exportation facilities. In our experiments we exchanged CAD data whose meta-data were described by both the STEP standards AP203, Configuration Controlled 3D Design of Mechanical Parts and Assemblies [15], and AP214, Core Data for Automotive Mechanical Design Processes [16]. The experiments were made using two of the most widespread CAD systems, namely, I-DEAS and Pro/ENGINEER. In particular, we used their modules for development of part geometry in the initial design of product parts. However, EMQ is general and can work with any EXPRESS information model and Part 21 data exchange file.

The EXPRESS data representation requires multiple inheritance which is supported by the data model and the query language of Amos II [29] on which EMQ is based. Queries retrieving all components in a hierarchical product structure as well as the model completion require transitive closure operations. The transitive closure facility of AmosQL allows for repeated application of a function with the same argument and result type until either no further application is possible, or a specified number of repetitions has been reached.

Furthermore, EMQ uses the mediator/wrapper approach [8, 9, 23, 32, 33] to allow queries and views that combine STEP/EXPRESS based data with other kinds of data, such as relational databases [7] and XML-based data [24]. These facilities rely on the techniques developed for OO mediation in Amos II [18, 19], a discussion of which is outside the scope of the present paper.

This paper first discusses related work. Section 3 describes the architecture of EMQ, while Section 4 provides certain examples of how EMQ queries are expressed and visualized. Section 5 describes the model completion implementation, and Section 6 concludes.

2. Related work

Early work on CAD databases [1, 4, 3, 10] concentrated on object storage rather than high-level queries and [22] discussed various approaches to the geometric data modeling. In contrast, our contribution is a query system for high-level user queries to complex standardized CAD data, where the system provides *model completion* of the query result for visualization by means of a CAD system. The management of engineering data in Amos II and querying using AmosQL is discussed in [26] with no consideration of problems involved in the visualization and completion of the query result.

EQL [20] is an EXPRESS query language intended for ad hoc queries on data in STEP/Part 21 data files. The language is not closed, i.e., the query result can not be queried and it has no geometric model completion. The aim of the EXPRESS-X language [12] and its predecessor BRIITY [11] is to translate data (e.g., Part 21 files) represented in one EXPRESS information model into the corresponding data in another, thereby reconciling heterogeneous and conflicting data. Its ability to construct views of EXPRESS data provides basic query facilities. EXPRESS-X is primarily intended to be a schema translation language and is not a general query language for CAD data. Unlike EMQ, it does not deal with the problem of the model completion of query results that is needed in order to make the result of an ad hoc query acceptable to a CAD system.

We have not intended to develop a new query language, but rather to extend an existing OO query language with certain required operations, such as geometric model completion.

[25] discusses certain approaches for optimizing a PDM system that resides on top of a relational DBMS. DIVE [21] is a database integration tool for virtual engineering applications providing for separate storage and management of product structure data (in a relational database) and spatial data (in ORDBMS). The system provides query facilities on both kinds of data, i.e., structural queries through CAD DB and advanced spatial queries, but it does not send query results to the CAD system for visualization. While we do not consider such special representations for spatial data, nor the implementation of spatial operations, we do address the querying of geometric and topological data concerning the shape and size of product parts in standardized object-oriented engineering schemas and representations. Whereas a spatial query result is a set of existing spatial objects, a query on OO representation of geometric and topological data may result in an object structure non-complete in a spatial sense. This type of queries therefore requires a completion algorithm for the query results.

Several systems have been developed for visualizing the results of database queries [5, 2]. Rather than connecting the visualization directly to the DBMS, we utilize a general CAD system for this purpose. This requires standardized data exchange between EMQ and the CAD, along with our model completion methods that modify the results of a query such that they can be visualized by the CAD system. In addition to visualization, this technique allows for the use of the query result in further analysis and in other operations

inside a CAD system.

3. EMQ Architecture

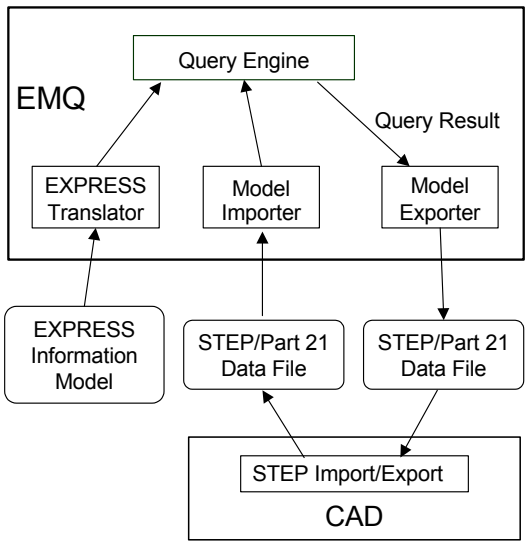

Figure 1. The EMQ architecture

illustrates the EMQ architecture. Since the data models of EXPRESS and Amos II differ, we need to translate meta-data from EXPRESS into Amos II. This is performed by the *EXPRESS Translator* module. This module reads any EXPRESS information model and translates it into a corresponding Amos II database schema. The translator carries out certain mappings between the data representations in EXPRESS and Amos II (for more details see http://www.csd.uu.se/~milena/Mapping.pdf). Since both EXPRESS and Amos II have OO data models, the mapping is straightforward.

Once an EXPRESS schema has been imported into EMQ, the *Model Importer* makes it possible for the system to access any Part 21 data exchange file using that schema generated by a CAD system. It creates new database objects and sets relationships between them in order to fully represent the CAD data in Amos II.

The OO query language AmosQL allows for general database queries over CAD data, and the result of such a query is a set of database objects. In order to export this result back into the CAD, the *Model Exporter* generates a new Part 21 exchange file representing the result of the query. This is sent back to the CAD system for visualization or further analysis.

Queries that result in geometric and topological data are of particular interest for visualization. However, encoding this type of query result as a Part 21 file is not sufficient for representing it in a CAD system since it must be enclosed by an object structure rooted in an object that represents a specific geometric model. We have therefore developed a generic *Model Completion* framework as a part of the model exporter. Additional objects are created within this framework which, together with the query result, form a CAD model that is a shape representation of a fictional product part. The architecture of the Model Exporter is illustrated in Figure 2.

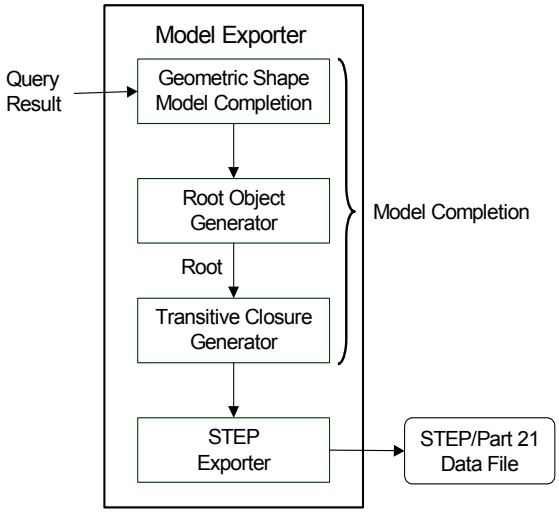

Figure 2. The Model Exporter

The Model Completion framework follows the STEP standard requirements for representing a product shape. First, *Geometric Shape Model Completion* (GSMC) performs the task of creating an enclosing structure for a given query result with an appropriate root object. GSMC deals with two kinds of problems, namely, different result sets need different root objects, and more than one type of root object is possible for certain result sets. This module is generic and table driven.

The root object of the enclosing structure must be one of the Geometric Shape Models (GSMs) defined in the STEP standard [13]. Two classic types of geometric modeling of solid objects are included in this standard: constructive solid geometry and boundary representation. The latter, for example, is presented by *Manifold_solid_brep* and its subtypes. Certain models in the standard, such as *Geometric_set* and *Shell_based_surface_model*, represent less complete descriptions of the geometry of a product. They allow for communication with systems whose capabilities differ from those of solid modeling systems. They are particularly useful for communicating the results of a DB query with a CAD system that accepts certain incomplete solid models.

The second step in the model completion framework is

executed by the *Root Object Generator (ROG)*. The ROG creates additional *root* objects whose roles are to place the GSM structure that has been created within a context, e.g., the dimensionality of the coordinate space and units of measurement. Together they form a CAD model acceptable to the CAD system. One of the objects plays the role of a root object for the entire model exported to the CAD.

The exported data set needs to be closed. For example, an object representing a surface needs to be exported together with the objects that represent points on this surface. The query result, the objects created by GSMC, the root objects, and all objects representing attributes of the query result are collected by the *Transitive Closure Generator* (TCG). They form the closure of the root object of the CAD model.

Finally, the data in the object set collected by TCG is encoded by *STEP Exporter* into a file in STEP/Part 21 standardized format that is sent to the CAD system. While the GSMC and the ROG modules depend on the EXPRESS information model for a particular application domain, the TCG and the STEP exporter are domain independent.

4. CAD data example queries

In general, AmosQL queries are expressed as
select <result>
from <type specification>
where <condition>

The *<type specification>* defines variables bound to the extent of types. The conditional expression in the where clause restricts the cartesian product of the type extents. The select clause specifies a result tuple that is calculated for every variable binding in the restricted cartesian product of the type extents.

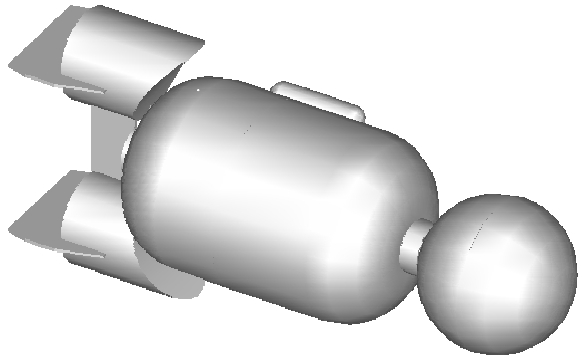

Figure 3. Space ship

Let us consider a simplified CAD model of a space ship, shown in Figure 3. We export the data in STEP/Part 21 data format from the CAD system into an EMQ database where the corresponding AP203 schema is already loaded. A subset of the schema relevant to the given example is shown in Figure 4 as an extended entity-relationship (EER) diagram. The relationships have a logical direction (i.e., they are *functional relationships*), but can still be queried in the inverse direction.

Although the query facilities of EMQ are general, we focus in the present discussion on queries over CAD data with geometric and topological results. EMQ can query and send to the CAD system the geometric representation of any component of an assembly. For example, the object structure that represents a component named "tail turbine" in Figure 5a is extracted by calling a predefined query function *get_component*[1]:

select get_component("s1_tail_turbine");

Given the name of a component, this function returns a *shape_representation* object (Figure 10) that is a root of the object structure defining the component geometry.

Data can be extracted by geometric condition in the query. For instance, in order to see all planar faces in the tail turbine component, we specify the following query that retrieves all faces in the tail turbine having an associated plane surface geometry[2]:

select fs from ST_face_surface fs, ST_plane pl
where face_geometry(fs) = pl and
fs =[3] *entclosure(get_component("s1_tail_turbine"));*

The *fs* and *pl* variables are bound to the extents of *ST_face_surface* and *ST_plane* types respectively. The first condition selects those *ST_face_surface* objects that are related to geometric surfaces of type *ST_plane* through the relationship *face_geometry*. The second condition restricts the selection to only those objects of type *ST_face_surface,* that occur in the component named "s1_tail_turbine". The *entclosure* function implements a transitive closure operation. Applied on the root object of type *ST_shape_representation*, it returns all objects in the structure defining the component geometry.

In order to import the result of the example query into a CAD system, the model exporter module of the EMQ is called, which is implemented as a system query function:

export_model (Bag of Entity query_result)

The function *export_model* creates a new complete CAD model representation for the result of an ad hoc CAD data query. For example:

export_model(select fs
from ST_face_surface fs, ST_plane pl
where face_geometry(fs) =pl and
fs= entclosure(get_component ("s1_tail_turbine")));

[1] *get_component* is defined through a rather complex query. It is omitted here for the sake of brevity.
[2] The names of the mapped EXPRESS types have the prefix '*ST_*' in order to avoid name collisions with other Amos II types.
[3] In our system '=' is overloaded to denote set membership when an operand denotes a set.

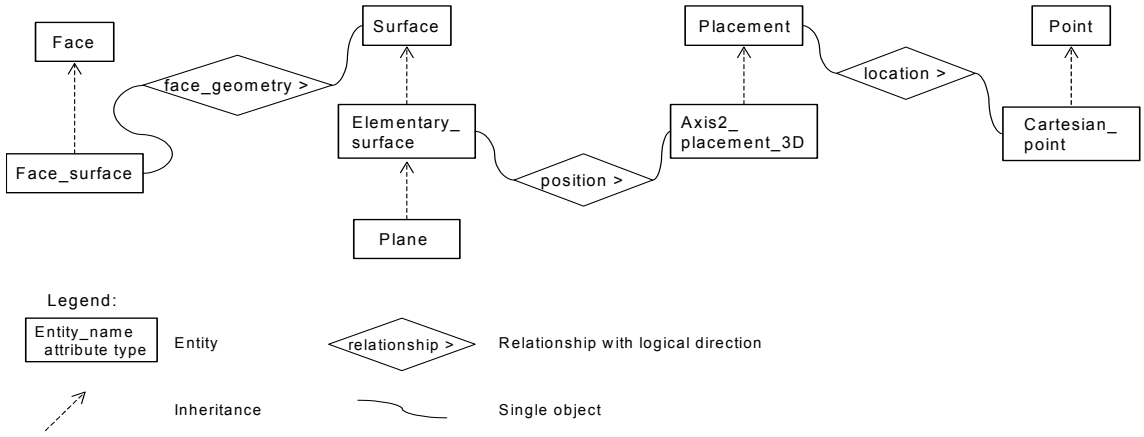

Figure 4EER diagram of subset of AP203 for the examples

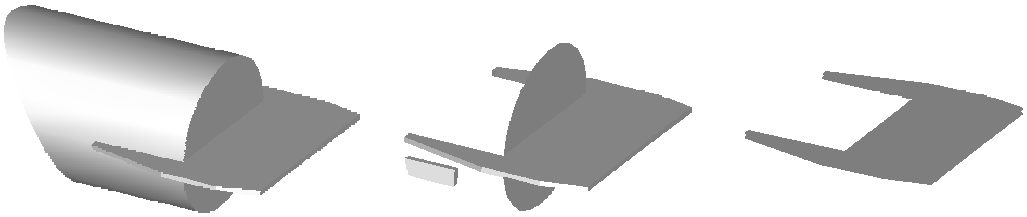

Figure 5 (a) Tail Turbine, (b) Plane faces in Tail Turbine, (c) Plane faces with more than 4 edges in Tail Turbine

The function first calls the GSMC module in order to find an appropriate geometric model for the query result type and construct an object structure that encloses the result set. The algorithm for GSMC is presented below in Section 5.

The system has information about the GSMs required to contain the query result. It automatically determines which of these models can be containers for the particular query result by using the relationships between types. Since more than one GSM is possible for certain result types, EMQ uses a system table that prioritizes the models in order to resolve ambiguities. In our example, data of type *Face_surface* can be contained in several geometric shape models and EMQ prioritizes the least limiting *Shell_based_surface_model*.

The user can override the automatic selection of the GSM by explicitly specifying the name of the model as an optional second argument of *export_model*:

export_model (Bag of Entity query_result,
Character geom_model)

After creating the GSM structure, the necessary root objects of the CAD model are added by the root object generator. The root of the model is passed to the transitive closure generator that collects all objects composing the complete CAD model. For the example query this means that *Plane* objects will be added to the model together with *Face_surface* objects because of the *face_geometry* relationship.

The collected object set is then sent to the STEP Exporter, which creates a Part 21 data exchange file encoding the model. The CAD system visualization of the model containing only the plane faces of the tail turbine component of the ship as selected by our query can be seen in Figure 5b.

In a similar way we can extract all faces with planar geometry and more than four edges in some of their boundaries from the tail turbine component (Figure 5c), combining in one query structural, geometrical, and topological criteria:

```
export_model( select fs
      from ST_face_surface fs, ST_plane pl,
ST_edge_loop l
where fs =
entclosure(get_component("s1_tail_turbine"))
and face_geometry(fs)= pl and
bound(bounds(fs)) = l and count(edge_list(l))>4);
```

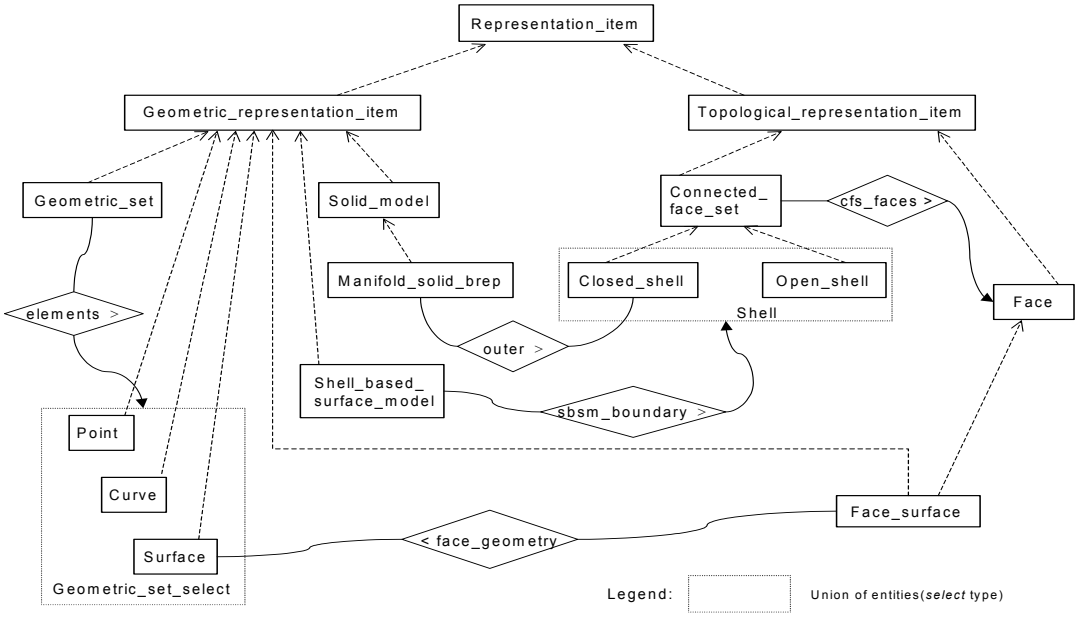

Figure 6. Subset of AP203 and AP214 illustrating model completion

5. Model Completion

In this section we will consider in detail the model completion framework and, in particular, the algorithm for geometric model completion.

5.1 Geometric Shape Model Completion

Geometric model completion is needed for geometrical and/or topological query results. STEP standards require this type of data to be enclosed by a structure rooted in some GSM, such as *Shell_based_surface_model, Geometric_set* and *Manifold_solid_brep* explained in Sec. 3 and illustrated in Figure 6. Different parts of the standard define different valid geometric models; for example, *Face_based_surface_model* is included in the AP214 schema, but is not used in AP203. Since the choice of the container for the query result is application dependent, we provide the system with information about the valid GSMs in a generic way through a system function called *priority table*:

valid_geom_models() -> Vector of Type t;

The system administrator shall set the function value to a vector (ordering) of all types representing GSMs for a particular standard schema ordered by how general the models are.

Since different GSMs are appropriate for different kinds of geometric and topological data, the work of GSMC depends on the type of the query result. The system needs knowledge of which GSM is appropriate for every kind of geometric and topological data. Following the relationships between types in the database schema, the system automatically determines which of the GSMs can be a root for the enclosing structure of the particular query result. The fact that more than one GSM is possible for certain result types is resolved through an ordering of the models by priority in the value of the *valid_geom_models()* function. GSMs that are less restrictive according to STEP standard constraints have a higher priority.

The example query on page 9 returns a collection of *Face_surface* objects. From the diagram in Figure 6 we see that possible GSMs for a *Face_surface* object are *Shell_based_surface_model* and *Manifold_solid_brep.* The system chooses the first one since it has fewer constraints and precedes the other in the priority table.
The system automatically finds a path through the schema from the GSM type determined to the query result type, creates instances of the types along this path, and connects these instances by setting functional relationships. In this way a new structure is created that contains the query result and has a root of GSM type. In our example, the system finds a path from *Shell_based_surface_model* through *Open_shell* to the result type *Face_surface*, creates one instance of each (except the result type), and connects the *Shell_based_surface_model* instance to the *Face_surface* result objects by setting the *sbsm_boundary* and *cfs_faces.*

There may at times be several paths when searching for one that connects the GSM and the query result type. In the example above there are ambiguous paths between *Shell_based_surface_model* and *Face_surface*, and the system does not know whether to choose the one through the *Closed_shell* type or the one through the *Open_shell* type. The GSMC therefore contains an *ambiguity table* as guidance for how to relate types when there are ambiguities.

amb_table(Type tstart, Type tend) -> Type t;

The ambiguity table specifies the next edge in the path, given the start and end types. In the simplified schema in Figure 6, the ambiguity table contains an element for the path from *Shell_based_surface_model* to *Face* type through the *Open_shell* type.

Queries returning objects of certain GSM types (*geometric_set*, *manifold_solid_brep*, etc.) do not require geometric completion. Thus, when the system automatically recognizes such types using the model priority table, it directly calls the ROG and TCG modules.

5.2 Algorithm for GSM completion of CAD data query

Figure 7 presents the main steps in the algorithm for the geometric model completion of a given query result.

The algorithm consists of three steps: determine the GSM type for the particular query result, search the path between the GSM type and the query result type, and construct the object structure enclosing the query result. The root of the constructed structure is then sent to the ROG module (section 5.4).

Input: set of database objects that are result of an ad hoc query.
Output: the root GSM object of the constructed object structure that encloses the query result.
Algorithm:
1. Analyze the query result to determine whether it needs a model completion. If model completion is needed:
1.1. The user has provided a GSM type as an argument. Either check its validity and continue with step 2, or raise an error message when the type is not a valid GSM;
1.2. The GSM is not provided. Run the algorithm in step 2 for one or more valid GSMs starting with the model with highest priority. Stop when an appropriate GSM has been found or when all models have been checked without success.
2. Search a path in the entity type graph between a valid GSM type and the query result type (we assume that the objects in the query result have the same type).
3. Construct an object structure with a root that is an instance of a GSM type using the path found in step 2.

Figure 7 The algorithm for the geometric shape model completion of the query result

The goal of the first step is to find a GSM type appropriate for the root of the enclosing object structure. The second step searches for a path between this type and the query result type. The algorithm used is a graph algorithm for an instance of the single-pair path problem [6] with additional modifications. The graph G = (V, E) has a set of vertices V that correspond to entity types in a given EXPRESS schema. The set of edges E includes the functional relationships between types in V. Since the functional relationships have logical direction, the edges in G are oriented:

$E = \{f_{t \to u}\}$, t, u$\in$V;

The graph is defined in a generic way in the system as follows:

- metaqueries to the metatype Type provide the definition of vertices V;
- the set of edges is modeled by a derived metafunction *contain_types*:

Contain_types(Type t)-> bag of <Type u, Function f>;

The function returns a pair <Type, Function> for each functional relationship defined on the argument type that connects it to other entity type, including functions whose result is an array or a union type since they can have an entity type as an element.

Modifications of the basic single-pair path algorithm are needed because of type inheritance and ambiguity when more than one path is possible. Let A and B be entity types for which a functional relationship is established in the schema: $f_{A \to B}$. The functional relationships are inherited: for each type A' subtype of A, the set of the functional relationships on A' includes $f_{A \to B}$ although f is defined only once - on type A. This property is automatically provided by the system through the inheritance mechanism. The functional relationship $f_{A \to B}$ is also a relationship to every type B' that is subtype of B, since every instance of B' is an instance of B. The inheritance of the function result type is not automatically provided and must be modeled by the algorithm or by the edge definition.

Another modification of the algorithm is needed because of the ambiguity. Since the algorithm should provide exactly one path for the construction in step 3, it needs tools for resolving ambiguity when more than one path exists for a given pair of vertices. We assume that different paths can have different values from the GSMC point of view. Instead of arbitrarily choosing one of the possible paths, the algorithm checks the ambiguity table to recommend a path between the current pair of vertices. This check is required only when more than one edge exits the current vertex and, therefore, the table contains

elements for only such pairs of vertices. The function *Recommended_path(Type t, Type q)* checks the contents of the ambiguity table for the current pair of vertices and returns a pair $< u, f_{t\to u} >$ of the preferred vertex u (type) and the edge (function) to it where the path should continue.

```
Input: current type and end type to which a path is
    looked for;
Output: boolean value True if a path exists and False
    otherwise. Global variable P contains the arcs in
    the path found;
Function:
boolean Search_path (t, end_type)
    S ← contain_types(t)
    if |S| = 0 return False
    for each s ∈ S, s = <u, f t→u>
        if (u = end_type) or Subtype(end_type, u)
        Append(P, <t, f t→u>)
        return True
    if |S|=1
        Extract element u from S
    else u ← Recommended_path(t, end_type)
        S1 ← S - {u}
    if Search_path(u, end_type)
        Append(P, <t, f t→u>)
        return True
    else for each u ∈ S1
        if Search_path(u, end_type)
            Append(P, <t, f t→u>)
            return True
    return False
```

Figure 8 Algorithm for recursive searching for a path between types in the entity type graph

In the pseudocode for searching the path in Figure 8, we assume that a global list variable P is defined in order to store pairs of kind $< t, f_{t\to u} >$ that define the arcs $f_{t\to u}$ in the path found. A new arc is appended at the end of the list (Append (P,x)) after previous successful recursive calls of Search_path function have already constructed the rest of the path to the end_type. As a result the list P starts with the arc leading to the end_type as needed by the next step.

The algorithm for the third step, presented in Figure 9, constructs an object structure that encloses the query result. The object structure is built in a bottom-up fashion. The list P is processed from the beginning using Pop(P) operation. For each pair $< t, f_{t\to u} >$ in the path P an instance of type t is created and the functional relationship $f_{t\to u}$ is set to the object of type u created in the previous step. The first "lowest" functional relationship is set to the query result objects. The output is the root of the constructed object structure, which is of the type specified in the first step GSM type. The algorithm presented here is simplified since the setting of the functional relationships of kind 1:1 and 1:M require special treatment.

```
Input: the query result and the path specification
    stored in the global variable P.
Output: the GSM root of the builded structure
    enclosing the query result.
Function:
GSMtype Create_model(query_result)
    e ←query_result
    while P ≠ 0
        <t,f> ← Pop(P)
        create o of type t
        f(o) ← e
        e ← o
    return e
```

Figure 9 Construction of the object structure enclosing the query result

In the current implementation, the analysis of the possible GSMs and the search for the path between types in the schema do not take into account the STEP standard facility for constraints that express restrictions on properties to ensure data validity and logical consistence. Through the GSM priority table and the ambiguity table we implement a way to provide the system with knowledge of how to work correctly, replacing information in the constraints. Constraints could be used to ignore certain formally incorrect geometric models. However, a large number of requirements are formulated only as *informal propositions* and the STEP standard does not provide formal algorithms (if such exist) to check them. For example, the requirement that the union of the domains of the faces that are contained in a *Connected_face_set* object should be connected is described only in the form of an informal proposition. Furthermore, the constraints provide restrictive but not constructive information for how to build a GSM (based on the fact that the construction is made always by a CAD or other engineering system). In addition, the experiments have shown that CAD systems accept (on different levels) models in which formal constraints and/or informal propositions are not satisfied.

It is thus impossible to assess the correctness of a constructed GSM structure if only STEP constraints are used, even if EMQ would have supported formal constraints. We instead currently rely only on the GSM priority table and the ambiguity table to automatically construct the model completion, which, as the experiments have shown, is accepted by the CAD systems.

5.3 Root Object Generator

A CAD model of a product shape requires specific types of the root objects defined in the standard. For example, both in AP203 and AP214 the main root object is an instance of type *Shape_representation* (Figure 10). As all GSM types are subtypes of the *Representation_item* (Figure 6), the GSM structure enclosing the query result has to be connected to the root object through the functional relationship *items*. Other attributes of the root object, such as the dimensionality of the coordinate space and measurement units, describe characteristics of the context in which the shape geometry is defined.

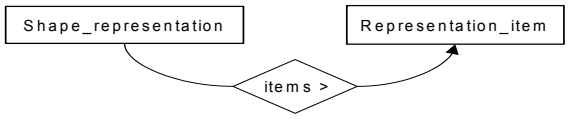

Figure. 10 Root object structure

This root object is in fact sufficient to construct a model that the experimental CAD systems import with warning messages about missing definitional data. In order to construct a model that is accepted by the CAD without any warnings, a wider set of root objects has been implemented as well. They describe a fictional product part and are beyond the present scope of discussion.

5.4 Transitive Closure Generator

Finally, the model has to be *closed*, which means that all assigned attributes of the exported objects must also be exported. This is a transitive closure operation that, according to the STEP/Part 21 encoding requirements, follows every functional relationship defined in the STEP/EXPRESS schema. The algorithm for the transitive closure is generic and does not depend on the query result or the particular EXPRESS schema.

Input parameter for the TOG is the root object created by the previous module. The TOG collects all objects that compose a complete model capable of being represented in the CAD system. This set includes the query result, GSMC objects, root objects, and all other objects reachable through functional relationships from these objects. For example, in our query the objects of type *Face_surface* require the objects of type *Surface* that are values of the functional relationship *face_geometry* to be included in the model. The latter objects in their turn require objects of type *Axis2_placement_3D* to be added as values of the functional relationship *position*, and *Cartesian_point* objects to be included as values of the *location* function. The object closure set is then passed to the STEP/Part-21 file exporter for encoding into an exchange file.

6. Conclusions and Future Work

The main results of the current work are the following:

An ad hoc OO query facility has been developed for data in CAD systems.

A query result can be exported to a CAD system using an ISO standardized format (STEP Part 21).

By means of the exportation facility the result of a query can be visualized or analyzed through a CAD system.

A model completion algorithm is implemented for extending a query result such that it becomes a model acceptable by a CAD system.

The mediator facilities permit queries combining CAD data with other types of data.

The model completion algorithm is driven by the STEP/EXPRESS information model complemented by two system tables. One prioritizes possible geometric shape models, and the other resolves ambiguous paths between types in the information model.

Possible future work includes:

Implementation of remaining EXPRESS features as functions, procedures, etc., needed for constraint checking. The constraints in STEP standards would complement but not replace our model completion system tables.

Investigation of the ability to specify queries with functions that make advanced calculations over engineering data. The foreign function mechanism of Amos II can be used for this purpose.

Enhancing performance by developing new data representations and indexing techniques for complex EXPRESS data structures.

7. References

[1] R. Ahmed and S.B. Navathe: Version Management of Composite Objects in CAD Databases, *SIGMOD'91*, 218-227, 1991.

[2] L. Bouganim, T.C-S. Ying, T-T. Dang-Ngoc, J-L. Darroux, G. Gardarin, and F. Sha: MIROWeb: Integrating Multiple Data Sources Trough Semistructured Data Types. In *Proc. of the 25th VLDB Conference,* Edinburgh, Scotland, 1999.

[3] A.P. Buchmann and C.P. de Celis: An Architecture and Data Model for CAD Databases, In *Proc. of the 11th VLDB Conference*, 105-114, 1985.

[4] D.S. Batory and W. Kim: Modeling Concepts for VLSI CAD Objects. *ACM Trans. on Database Systems,* 10(3):322-346, 1985.

[5] M. Carey, L. Haas, V. Maganty, and J.Williams: PESTO: An Integrated Query/Browser for Object Databases. In *Proc. of the 22nd VLDB Conference,* pages 203-214, Mumbai, India, 1996.

[6] T. Cormen, C. Leiserson, R. Rivest, *Introduction to Algorithms*, The MIT Press, 1990.

[7] G. Fahl and T. Risch: Query Processing over Object Views of Relational Data*, The VLDB Journal*, 6 (4): 261-281, 1997.

[8] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y.Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems (JIIS),* Kluwer, 8(2): 117-132, 1997.

[9] L. Haas, D. Kossmann, E.L. Wimmers, and J. Yang: Optimizing Queries across Diverse Data Sources. In *Proc. of the 23rd VLDB Conference,* pages 276-285, Athens, Greece, 1997.

[10] M. Hardwick and D. Spooner: The ROSE Data Manager: Using Object Technology to Support Interactive Engineering Applications, *IEEE Trans. on Knowledge and Data Engineering*, 1(2): 285-289, 1989.

[11] T. Härder, G. Sauter, and J. Thomas: The intrinsic problems of structural heterogeneity and an approach to their solution. *The VLDB Journal,* 8: 25-43, 1999.

[12] International Organization for Standardization:. *ISO 10303-14, Industrial automation systems and integration - Product data representation and exchange- Part 14: Description methods: The EXPRESS-X Language Reference Manual.* ISO Document TC 184/SC4/WG11 N117, 2000. http://www.nist.gov/sc4/step/parts/part014/CD/doc/

[13] International Organization for Standardization: *ISO 10303-42:2000, Industrial automation systems and integration - Product data representation and exchange - Part 42: Integrated generic resources: Geometric and topological representation*. Second Edition. ISO Document TC 184/SC4/ WG12 N 540, 1994. http://www.nist.gov/sc4/step/parts/part042e2/is/n540/

[14] International Organization for Standardization: *ISO 10303-1:1994, Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles*, 1994.

[15] International Organization for Standardization: *ISO 10303-203:1994, Industrial automation systems and integration — Product data representation and exchange — Part 203: Application protocol: Configuration controlled 3D design of mechanical parts and assemblies*, 1994.

[16] International Organization for Standardization: *ISO 10303-214:1998, Industrial automation systems and integration - Product data representation and exchange - Part 214: Core Data for Automotive Mechanical Design Processes*. ISO Document TC 184/SC4/WG3, 1996.

[17] International Organization for Standardization: *ISO/DIS 10303-21:1999, Industrial automation systems and integration - Product data representation and exchange - Part 21*: *Clear text encoding of the exchange structure*. ISO Document TC 184/SC4/WG11 N102, 1994.

[18] V. Josifovski and T. Risch: Functional Query Optimization over Object-Oriented Views for Data Integration, *Journal of Intelligent Information Systems (JIIS),* 12(2-3), 1999.

[19] V. Josifovski and T. Risch: Integrating Heterogeneous Overlapping Databases through Object-Oriented Transformations, In *Proc. of the 25th VLDB Conference,* Edinburgh, Scotland, 1999.

[20] D. Koonce, L. Huang, and R. Judd: EQL an EXPRESS Query Language*, Computers and Industrial Engineering*, 35(1-2),1998.

[21] H.P. Kriegel, A. Muller, M. Pötke, and T. Seidl: DIVE: Database Integration for Virtual Engineering, *Demo session on 17th International Conference on Data Engineering*, Heidelberg, Germany, 2001.

[22] A. Kemper and M. Wallrath: An Analysis of Geometric Modeling in Database Systems, *ACM Computing Surveys*, 19(1), 1987.

[23] L.Liu and C. Pu: An Adaptive Object-Oriented Approach to Integration and Access of Heterogeneous Information Sources. *Distributed and Parallel Databases,* Kluwer, 5(2), 167-205, 1997.

[24] H. Lin, T. Risch, and T. Katchaounov: Adaptive data mediation over XML data. To be published in special issue on "Web Information Systems Applications" of *Journal of Applied System Studies (JASS)*, Cambridge International Science Publishing, 2002.

[25] E. Muller, P. Dadam, J. Enderle, and M. Feltes: Tuning an SQL-Based PDM System in a Worldwide Client/Server Environment. In *Proc. of the 17th International Conference on Data Engineering*, Heidelberg, Germany, 2001.

[26] K. Orsborn: Management of Product Data Using an Extensible Object-Oriented Query Language, *International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, Phoenix, Arizona, USA, 1996.

[27] PTC Corporation: Pro/ENGINEER, http://www.ptc.com/products/proe/index.htm, 2001.

[28] T. Risch and V. Josifovski: Distributed Data Integration by Object-Oriented Mediator Servers, *Concurrency and Computation: Practice and Experience J.,* 13(11), John Wiley & Sons, 2001.

[29] T. Risch, V. Josifovski, and T. Katchaounov: *AMOS II Concepts*, Department of Information Science, Uppsala University, 2000, http://www.dis.uu.se/~udbl/amos/.

[30] SDRC Corporation: *Introducing I-DEAS*. http://www.sdrc.com/ideas/index.html, 2001

[31] D. Schenck and P. Wilson: *Information Modeling: The EXPRESS Way*. Oxford University Press, 1994.

[32] A. Tomasic, L. Raschid, and P. Valduriez: Scaling Access to Heterogeneous Data Sources with DISCO. *IEEE Transactions on Knowledge and Date Engineering*, 10(5), 808-823, 1998.

[33] G. Wiederhold: Mediators in the architecture of future information systems, *IEEE Computer,* 25(3), 38–49, 1992.