RELATIONAL DB IMPLEMENTATION OF STEP BASED PRODUCT MODEL

SEOK-JOON YOU, DONGHOON YANG AND CHARLES M. EASTMAN
College of Architecture Ph.D. Program
Georgia Institute of Technology, Atlanta GA

A**bstract.** We present the implementation of GTCIS2SQL a relational database implementation of CIS/2, an industry-led product model for the structural steel used in building construction. GTCIS2SQL supports web-based communication with applications exchanging data though P-21 files for both input and output. It supports a variety of query capabilities. Details of the implementation are presented.

INTRODUCTION

For more than a decade, STEP (ISO 10303) and its underpinning technology have been developed and deployed to various engineering design and production areas. For the AEC (Architecture, Engineering and Construction) industry, a small but growing set of product models that use the STEP technologies support an increasing number of AEC software products (e.g. IFC and CIS/2).

As the use of STEP technology becomes more available to end-users, the issues of access, ease of use and coordination management are becoming more important. Like other engineering fields, an AEC project (e.g. design and construction) involves various tasks such as design, structural analysis, detailing, and so forth. This means that the information (specification) of a same part is stored into multiple files each of which is generated by different software. Such redundancy can lead to inconsistency among the representations of the part unless they are managed carefully.

There are many potential advantages if the STEP data is stored and managed by a dedicated database management system (DBMS) rather than a conventional file system. Some of them are listed below:

- The system can reduce the inconsistency of its data because all information regarding a specific part of the design resides in a

single place (the repository) rather than multiple files; propagation of changes among different views is made much more controlled and easier.
- Sharing of the same STEP data among multiple applications helps closer cooperation among the users.
- The data management system allows monitoring of current project status, as the information is accessible any time, on demand.
- Data management is potentially facilitated by reducing the unit of management to the object level, rather than the file.

Two different types of the DBMSs are available on the market for managing the STEP product model data: Object-Oriented Database Management System (OODBMS) and Relational Database Management System (RDBMS). Previous research (Loffredo92 and Snyder et al.95) claimed that OODBMS usually performs better than RDBMS. A RDBMS, however, has a wide variety of implementations to run on, is easily moved between different vendor's systems, is commonly available in engineering and construction organizations for use in accounting and ERP systems, have extensive interfaces, and are well known to the IT community.

This paper presents the implementation details of GTCIS2SQL, a Relational database implementation of a STEP-technology product model repository. It is able to support the implementation of any EXPRESS language schema, although it has been primarily implemented to support CIMsteel Version 2 (CIS/2), an industry-led product model initiative for modeling structural steel throughout its building and construction life cycle. GTCIS2SQL supports input and output communication with steel applications in the Part-21 encoded text file format (ISO 10303-21) and supports a limited range of queries. The research project presented in this paper uses SQL Server 2000, a commercial Relational DBMS running under Windows NT/2000/2003 server operating systems. The paper presents the mapping methods used in the implementation, its management and query capabilities and implementation.

GENERAL STRATEGY

EXPRESS, a standard STEP schema definition language (ISO10303-11), has the following characteristics that need to be considered for database implementation.

- EXPESS is an object-flavored data modeling language, encouraging deep inheritance structures

- It offers some challenging syntactic constructs, including SELECT types that are execution-time type selectors and powerful multiple inheritance constructs (AND and ANDOR inheritance constraints)
- Lacks encapsulation or other strong forms for depicting data dependency
- Includes a rich set of constructors, including sets, bags, variable length lists, and fixed set arrays.

Since it is a data modeling language and not a programming or database language, an EXPRESS schema must be translated into appropriate implementation. Such translation can be done through either direct programming language binding (early binding), or interface to a data dictionary (late binding). There are several toolkits that provide the language binding (i.e. they automatically generate class definitions from EXPRESS) and/or create data dictionaries with access functions to them.

The data model also needs to be translated into an appropriate storage format. From a historical view of data-mapping, translation of a data model (or schema) are not an oddity to relational DB modelers: many data modeling languages, such as Entity-Relationship models NIAM, IDEF1x and others have been widely used for designing Relational databases. Translation of object-oriented programming language classes such as Java classes into Relational databases are also described in Ambler (1998).

Translation of the EXPRESS schema is similar because the modeling language adopted its key ideas from object-orientation paradigm; however, there are several differences among them, which are listed as follows:

- In terms of the object design, Object-Oriented Programming Language (OOPL) objects has more strength on their 'behavior'-attributes are meaningful only when the objects are operational (i.e. the objects are 'running' as programmed); in STEP models, the 'attribute' and 'relationship' of its entities are of primary interest.
- For STEP models, behaviors (or member functions) of an object are not necessarily stored in the database; some persistent OOPL objects (especially Java objects, which are executed on a common 'virtual machine') also have behaviors stored in the database.
- STEP models usually have far more entities than typical OOPL projects. For a development of a single object-oriented program, less than 20 classes are desirable because the programmer has difficulty in managing its source code if there are more classes

than that. However, the scope of a STEP model goes beyond a single software. For typical STEP models, it is common to have hundreds of entities.

STEP Schema	Number of entities in this schema
AP 201 Explicit drafting	220
AP 202 Associative drafting	347
AP 203 (TC2) Configuration-controlled design	251
AP 210 Electronic assembly, interconnection & packaging design	649
AP 214 (FDIS) Core data for automotive mech design processes	891
CIS/2 (LPM 6) Structural steel	731
IFC (2X)	370

Table 1 number of entities in EXPRESS schemas

Unlike programming languages, there is no standard implementation method for translating STEP model into the relational data model, although such translation is considered to be feasible and some implementation cases can be found Loffredo (1998) and Morris (1990).

TRANSLATION METHODS

STEP database systems have been built by various research groups and commercial software companies. They can be categorized according to the type of the DBMS they used and mapping methods:

- With respect to the backend DBMS:
  - **Relational DBMS**: GTCIS2SQL (Georgia Tech), Adachi (2002), JSDAI (LKSoft GmbH)
  - **Object-oriented DBMS**: Loffredo (1998), ST-ObjectStore (STEP Tools, Inc.)
  - **In-house persistent data storage system**: ST-Developer (STEP Tools, Inc.), EXPRESS Data Manager (EPM Technology AS)

- With respect to the mapping methods:
    - Mapping each STEP **entity** (class) into corresponding database tables- all entity attributes are stored in a same table as tuples: (GTCIS2SQL and Adachi's system)
    - Mapping each STEP **data type** into tables. All attributes of the same data type will be stored in a same table, while attributes of a same entity will be distributed to different tables regarding their data types.

- With respect to the programming language binding style
    - Direct language binding (early-binding , as defined in STEP Part 20-series) using built-in persistence OOPL classes: Loffredo (1998)
    - Use of data dictionary and function calls to access it (i.e. late-binding)

ST-Developer (STEP Tools, Inc.) and EXPRESS Data Manager (EPM Technology AS) are the subsystems of general-purpose STEP toolkits. The developers claim that their in-house database systems perform better than other commercial DBMS; however, they are excluded from our choice because they lack two functionalities: advanced query functionality and transaction management in a multi-user environment, including concurrency control.

Object-oriented database implementations different database technology and commercial data repository systems are not our consideration. OODBMS do not provide seamless access through SQL nor do commercial data repositories. RDBMS by VTT is limited to IFC, does not support multiple supertypes and ANDOR subtype, which is required for CIS/2 database and other ISO-STEP APs. LKSOFT is also commercial. NIST approach does not support population of non-leaf node entity, which is required in CIS/2 and other EXPRESS models. For example, some of the new entities in revised CIS/2 LPM/6 are specialization of existing LPM/5 entities, which are not leaf node entities anymore.

LANGUAGE BINDING STYLE

Standard Data Access Interface (SDAI, ISO10303-22, ISO 10303-23, 24, and 27) allows these two different interface styles for programming languages. We chose ISO 10303-24 (C language binding) over ISO 10303-23 (C++ language binding). As shown in Table 1, the numbers of entities in STEP models are hundreds, which makes programming very difficult when

they are directly translated into C++ classes, which is less feasible for adding new member functions to them unless the EXPRESS entities have a single base entity class.

Mapping EXPRESS model to relational model

Schema mapping method for object oriented to relational has several variations as shown in Figure 1. EXPRESS-G diagram in the figure is a part of company schema, adopted from Fahl (1997). Relational modeling alternatives are listed underneath.

Mapping alternative (R1) separates attribute to each entity type. (R1) makes largest number of mapping relations among the alternatives considered, which makes hard to manage all the relations efficiently. (R1) stores attributes of different entity types in the same relation. Attributes of 'secretary' and 'salesman' are stored in 'employee' entity relation. If there can be instance of 'employee', 'employee' relation can store instance of 'employee', 'secretary' and 'salesman'. Alternative (R2) duplicates all the supertype attributes in the subtype. (R2) separates attributes by entity type, so one relation carries an entity type and its attribute values. Alternative (R3) includes every inherited entity type in one relation. The jobtype attribute is used to specify entity type. (R3) allows empty attributes, which are not applicable to populated subtype. (R2) and (R3) may reduce the number of relation when instantiation is allowed only for 'leaf node' entity type, and when ANDOR relation does not exist. (R2) and (R3) also reduce 'JOIN' with other relations in retrieving information, since it includes all supertype attribute in one relation. Alternative (R4) is based on attribute type rather than entity type, which is quite distinctive from other alternatives. Number of relation is reduced to the number of basic attribute types, which is relatively quite small to the number of entity type.

Most object orient model to relational model mapping methods use one of the alternatives presented here, depending upon the schema and purpose. Morris (1999) and Adachi (2002) uses alternative (R2). Fahl (1997) uses a variation of alternative (R1). GTCIS2SQL uses alternative (R1). LKSoft's JSDAI® SQL uses alternative (R4).

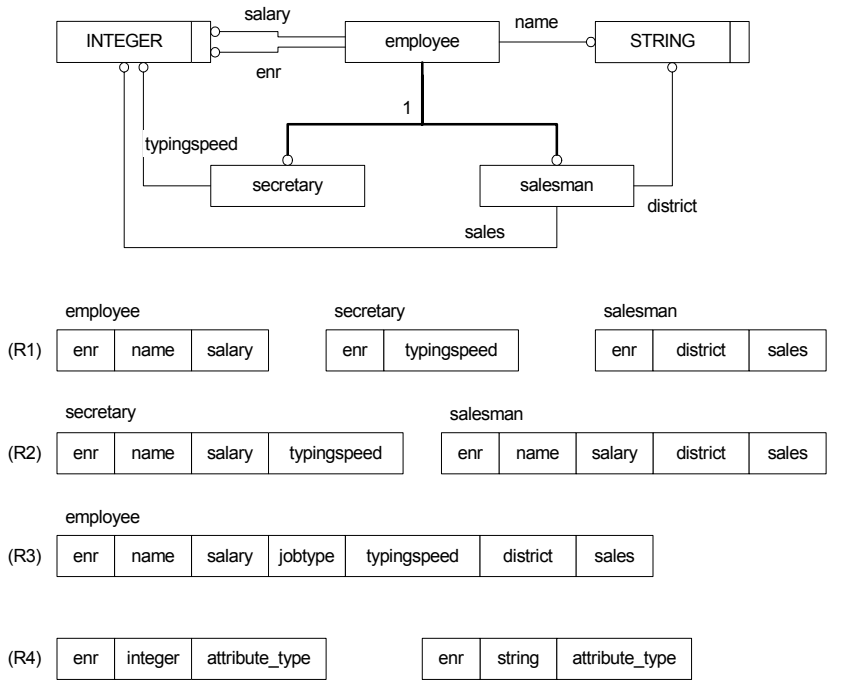

Figure 1: Mapping alternatives between the object oriented model and the relational model

VTT Building and Transport (Adachi 2002) developed an IFC (Industry Foundation Classes) applicable XML-based data repository (IFC model server). IFC is an integrated building model driven by IAI (international Alliance for Interoperability). VTT aims at:

1. storing IFC model data in a database system(SQL Sever, Oracle etc),
2. partial model data selection and merging function.
3. providing model server function as Web Services with SOAP communication between model server and client software. One of IFC's characteristic is that the model does not have ANDOR subtype and multiple inheritance, which allow to use mapping alternative (R2). Morris (1999) mapping method uses similar mapping method. She generated only end node tables in the assumption that instance is only populated for leaf node entity. CIS/2 LPM6 schema innately allows instantiation of non-leaf node entities.

GTCIS2SQL uses entity-to-table method (R1) rather than type-to-table method (R4). The two mapping methods have their own strengths and drawbacks. However, we chose the first method for our system for the

following reasons: entity-to-table mapping is considered simpler to implement because an entity and its attributes can be mapped into a corresponding table; in contrast, in type-to-table mapping entity and its attributes are stored in different tables so that additional mapping rules have to be specified to assemble dispersed attributes into an entity, or vice-versa.

GTCIS2SQL SCHEMA TRANSLATION RULES

The input to the GTCIS2SQL schema translator is any valid EXPRESS schema definition, which will then be translated into SQL table creation commands according to the translation rules that will be described in this section.

EXPRESS	SQL

ENTITY *identifier_1*
  SUPERTYPE OF (**ONEOF** (*identifier_2, identifier_3, …, identifier_n*)) ;
END_ENTITY:
 	
ENTITY *identifier_2*
  SUBTYPE OF (*identifier_1*) ;
END_ENTITY;
 	
ENTITY *identifier_3*
  SUBTYPE OF (*identifier_1*) ;
END_ENTITY;
…
ENTITY *identifier_n*
  SUBTYPE OF (*identifier_1*);
END_ENTITY;
	CREATE TABLE *identifier_1*
	(
	  tuple_id **BigInt PRIMARY KEY NOT NULL**
	)
	
	CREATE TABLE *identifier_2*
	(
	  tuple_id **BigInt PRIMARY KEY NOT NULL**
	)
	  ……
	CREATE TABLE *identifier_n*
	(
	  tuple_id **BigInt PRIMARY KEY NOT NULL**
	)

ENTITY *identifier_1*
  SUPERTYPE OF (identifier_2 **ANDOR** *identifier_3*)
END_ENTITY:
 	
ENTITY *identifier_2*
  SUBTYPE OF (*identifier_1*) ;
END_ENTITY;
 	
ENTITY *identifier_3*
  SUBTYPE OF (*identifier_1*) ;
END_ENTITY;	

Table 2 Translation rule for Entity type with subtypes

Translation of the EXPRESS entity and its hierarchy

For mapping entity hierarchies, GTCIS2SQL uses (R1) shown in Figure 1; if there are more than one sub-types in the entity hierarchy, they are further

decomposed according to the 'binding' mechanism, which 'glues' the partial subtypes into a single entity description containing all inherited attributes. An STEP instance's attributes are stored into individual tables according to the hierarchy, and then they can be queried by the same primary key.

Translation of the EXPRESS entity attributes

Translation of EXPRESS Attributes are done by one-to-one mapping, as most EXPRESS attribute types including defined types are reduced to basic data types. However, translation of aggregation types requires more translation steps than one-to-one mapping.

EXPRESS Aggregation types such as arrays and lists cannot be stored in one tuple because traditional relational data models don't have such data types. If the class has a fixed number of elements (e.g. an array), they can be translated as a series of attributes in a tuple. If the number of elements is indefinite (e.g. a list), however, we cannot store them in a single tuple and a separate table is required. In addition, EXPRESS aggregation class can also be nested indefinitely.

GTCIS2SQL translates aggregation classes as tables external to their owner entity, as shown in Table 3. The detailed translation rules are as follows:

- The table name is a combination of the entity name and the attribute name, in order to ensure that the table name is unique.
- Each tuple in the table has a foreign key *fk_owner*, which refers to the primary key of the entity instance that has the aggregation attribute; on the other hand, the 'owner' instance will not have the foreign keys.
- The *subscript* field, which has the ordinal value of the element, is provided for ordered lists such as arrays and lists.
- Nested aggregation will have multiple numbers of tables until they resolve the nesting. The name of the nested aggregation table will have '*_HAS_AGGR*' that is added to the table name of the upper level aggregation. This will make each table name unique.

EXPRESS	SQL
ENTITY *identifier*; *attribute* : *basic_type*; **END_ENTITY**;	**CREATE TABLE** *identifier* (tuple_id BigInt **PRIMARY KEY NOT NULL**, *attribute basic_type,* TY_*attribute* VarChar (128)) • TY_*attribute* will hold the text description of the exact basic type

ENTITY *identifier* ; *attribute* : **ARRAY** [*m*:*n*] **OF** *basic_type*; **END_ENTITY**; * *m, n* := *integer_literal* | *expression* * *n* must be larger than *m*. * unbound arrays are not allowed in EXPRESS.	**CREATE TABLE** *identifier* (tuple_id **BigInt PRIMARY KEY NOT NULL,**) **CREATE TABLE** *identifier*_HAS_*attrubute* (tuple_id **BigInt PRIMARY KEY NOT NULL**, fk_owner **BigInt**, subscript **Int**, value *basic_type***,** TY_value **VarChar** (128)) * in 'fk_owner', the prefix 'fk_' stands for 'foreign key'.
ENTITY *identifier* ; attribute : { **LIST** | **ARRAY** | **SET** | **BAG** [*m* : *n*] **OF** } *basic_type*; **END_ENTITY**;	**CREATE TABLE** *identifier* (tuple_id **BigInt PRIMARY KEY NOT NULL**, fk_owner **BigInt**) **CREATE TABLE** *identifier*_HAS_AGGR (tuple_id **BigInt PRIMARY KEY NOT NULL**, fk_**owner BigInt**) …. **CREATE TABLE** *identifier*{_HAS_AGGR}_HAS_AGGR_HAS_AGGR (tuple_id **BigInt PRIMARY KEY NOT NULL**, fk_owner **BigInt**, value sql_basic_type, TY_value **VarChar** (128)) * in 'fk_owner', the prefix 'fk_' stands for 'foreign key'. * Foreign key constraints are optionally removable. * 'NOT NULL' constraints are optionally removable.

Table 3 Translation rules for EXPRESS attributes

Translation of SELECT types

SELECT types are difficult to translate because this can be any of an enumerated set of STEP types, since a field (attribute) in a tuple is invariant in terms of its type.

GTCIS2SQL uses proprietary data type specific to Microsoft SQL Server: *SQL_Variant*. Although this type is not an ANSI standard SQL data type, other commercial DBMS also provide similar data types, which allows porting GTCIS2SQL to other platforms with minimal code change.

SQL_Variant type field is augmented by 'type information' fields. In Table 4, two type information fields are shown: *SL_attribute* field will have the name of the selected type, which will be one of *identifier_1* to *identifier_n*. *TY_attribute* field will have the resolved type name, which will be one of the basic EXPRESS types.

EXPRESS	SQL

ENTITY *identifier* ;
  attribute : *named_type*;
END_ENTITY;
 	
TYPE *named_type* = **SELECT**
(identifier_1, identifier_2, …,
Identifier_n);
END_TYPE;
	CREATE TABLE *identifier*
	(
	  tuple_id BigInt **PRIMARY KEY NOT NULL**,
	  attribute **Sql_Variant** /* SELECT type */,
	  SL_*attribute* **VarChar** (128),
	  TY_*attribute* **VarChar** (128)
	)

Table 4 Translation rule for SELECT types

Translation of EXPRESS constraints, rules, and derived attributes

There are basically two ways of translating WHERE clause constraints and rules: to translate into stored-procedures which will be stored in the database, or delegate such operation to the application which will import the data from the database. The GTCIS2SQL does not translate the rules because most STEP application requires toolkits such as ST-Developer, and most toolkits are able to evaluate the rules on-the-fly with their own libraries.

Basically, derived values also are treated in the same manner that is used for the rules, as they are calculated on demand and they are never persistent. In GTCIS2SQL, they are not translated into stored procedures.

Data management in a project context

From its purpose, a STEP model data is shared among project participants; what may happen is, different software applications can update the identical data then write to the database asynchronously. In order to manage the data within such environment, at least two information fields are required: (1) identifier of each data element (i.e. entity instance) can be identified, and (2) ID of the application, the computer, and the user that made the update. In addition, in order to store data from multiple projects, each instance should be distinguished by the projects so that the system can query data of a specific project.

STEP models such as CIS/2 and IFC define such information in different ways, but GTCIS2SQL system manages it within a unified framework independent to the model's data structure. A system-generated project information table stores the projects, which is referenced by the instances according to the project they are originated from. Another system-generated table stores mapping between globally unique instance identifiers and primary keys of the translated tuples. Tables for transaction history (time

stamp, type of the update, user information who made the update), which addresses (2) in previous paragraph, is being developed at the point of writing. These system-generated tables are independent of specific STEP models; mapping between the table and the model happens in transaction time.

Summary of the GTCIS2SQL schema translation rules

The overview of the schema translation rules are summarized as follows:

- Every EXPRESS entity type will correspond to a SQL table. Each EXPRESS entity instance will correspond to a unique table tuple (i.e. a record).
- If a given entity is a subtype of another entity, its super type attribute will be stored in the super type rather than in the subtype. The attributes from a same STEP entity instance, which are dispersed hierarchically into multiple database tables, will use the same primary key.
- For entity attributes, each EXPRESS basic type (e.g. number, real, logical, Boolean. text string) will be translated into corresponding SQL data type. The EXPRESS defined types will be resolved into the basic SQL data types, or they will be resolved into foreign keys if they are the references to other entities.
- EXPRESS Aggregation classes will be translated into separate tables, where each aggregation attribute will have a unique table. Thus each tuple will correspond to an element in the aggregation.
- Select type will be translated to SQL_Variant, which is dedicated to MS-SQL; Other DBMS provides similar types in different names.

INSTANCE TRANSLATION INTO GTCIS2SQL

Translation rules for instances

In GTCIS2SQL, Application data are stored to the database in two different manners. First, the application can translate their native data into SQL statements according to the translated schema. Second, application can create STEP Part 21 Clear-text encoding file, which can be passed to GTCIS2SQL.

As GTCIS2SQL has translation rules for arbitrary EXPRESS schemas, the system is able to read STEP instances (in STEP Part 21 format) and store

them to the relational DBMS, which is made possible by translation rules hard-coded to the system.

The translation rules for the instances are derived from the schema translation rules; once schema-translation rules are defined, instance translation is invariant. Table 5 shows the example how STEP instances are translated to SQL statements.

Translation operations

The translation operation is implementation of the translation rules. In addition to the translation, the result must be committed to the database.

To implement the GTCIS2SQL translation rules, the programming codes require lookup of the schema definition, because the instance file does not give complete information that is needed. For example, the schema translation rules shown in Table 2 require the name of the entity's super types, which are not given with the instances seen in Table 5.

STEP (Part 21- Clear text encoding)	**SQL**

`#10=IDENTIFIER_2();`
`#11=IDENTIFIER_3();`
	`INSERT INTO identifier_1 (tuple_id)`
	`VALUES (1)`
	`INSERT INTO identifier_2 (tuple_id)`
	`VALUES (1)`
	`INSERT INTO identifier_1 (tuple_id)`
	`VALUES (2)`
	`INSERT INTO identifier_3 (tuple_id)`
	`VALUES (2)`

`#10=(IDENTIFIER_1()IDENTI`
`FIER_2()IDENTIFIER_3());`
`#11=IDENTIFIER_2();`
`#12=IDENTIFIER_3();`
	`INSERT INTO identifier_1 (tuple_id)`
	`VALUES (1)`
	`INSERT INTO identifier_2 (tuple_id)`
	`VALUES (1)`
	`INSERT INTO identifier_3 (tuple_id)`
	`VALUES (1)`
	
	`INSERT INTO identifier_1 (tuple_id)`
	`VALUES (2)`
	`INSERT INTO identifier_2 (tuple_id)`
	`VALUES (2)`
	
	`INSERT INTO identifier_1 (tuple_id)`
	`VALUES (3)`
	`INSERT INTO identifier_3 (tuple_id)`
	`VALUES (3)`

Table 5 Translation example of a STEP instance using the rules shown in Table 2

QUERYING INSTANCE DATA FROM GTCIS2SQL

Query instance data is to access data stored in the GTCIS2SQL database. SQL (Structured Query Language) is a standard DML(Direct Manipulation Language) in relational database. SQL can be written by a user who wants to access the database. However, the user should be familiar with the schema mapping and the schema itself to manipulate the database. Through this access, the user will have fast access and total control over the database. The other way to access database is to use system-generated SQL. System generates SQL for a specific instance type as well as the data on the fly. Response time is slower than user-generated SQL but the user does not need the knowledge regarding schema mapping to database.

GTCIS2SQL provides both user-generated SQL and system-generated SQL. Making a user generated SQL is same as the writing SQL code on the premise that the user knows the schema and the mapping method as well as the table and the fieldname. The following query mechanism describes how we made the system-generated SQL and its implementation.

Query mechanism

Some of the information including supertype/subtype relations could be defined in the database side. However not every schema definition is stored in the database. The database does not include more than the data. As a result, the system generated SQL relies on a schema data dictionary to restructure instance data. The role of the data dictionary is to provide EXPRESS schema structure.

As mentioned, GTCIS2SQL database is structured according to the schema and the mapping rules. So restructuring instance requires referring to the schema structure, which has the data type definition as well as the mapping rules, which decide EXPRESS data type and SQL data type mapping. The mapping rule is the reverse of the schema translation rule. The data dictionary can be any type of structure, including relational database, object oriented database, data structure or any developer's program. GTCIS2SQL uses a variation of SDAI and SDAI data dictionary.

The data dictionary contains the schema definition including attribute name, attribute type, user-defined type, entity name, and supertype/subtype relation. GTCIS2SQL query generator structures SQL query with subtype-supertype information based on the data dictionary. A query for an entity converts supertypes and the entity names into table names and converts attribute names into field names of the tables according to the mapping rules.

In EXPRESS, an attribute of an entity type can reference any subtype of the attribute's entity type. The same attribute of an entity can be any one of the subtypes. In Figure 2, attribute 'installed_card' of 'desktop' instance can be either 'video_card' or 'sound_card'. Query for EXPRESS entity in the GTCIS2SQL mapping needs to be defined during runtime, since the entity type and the query are different by populated instance type. This makes a query for an entity itself and a query for an entity as an attribute different. The query for an entity itself is called 'first query' and the query for entity attribute is called 'subsequent query'.

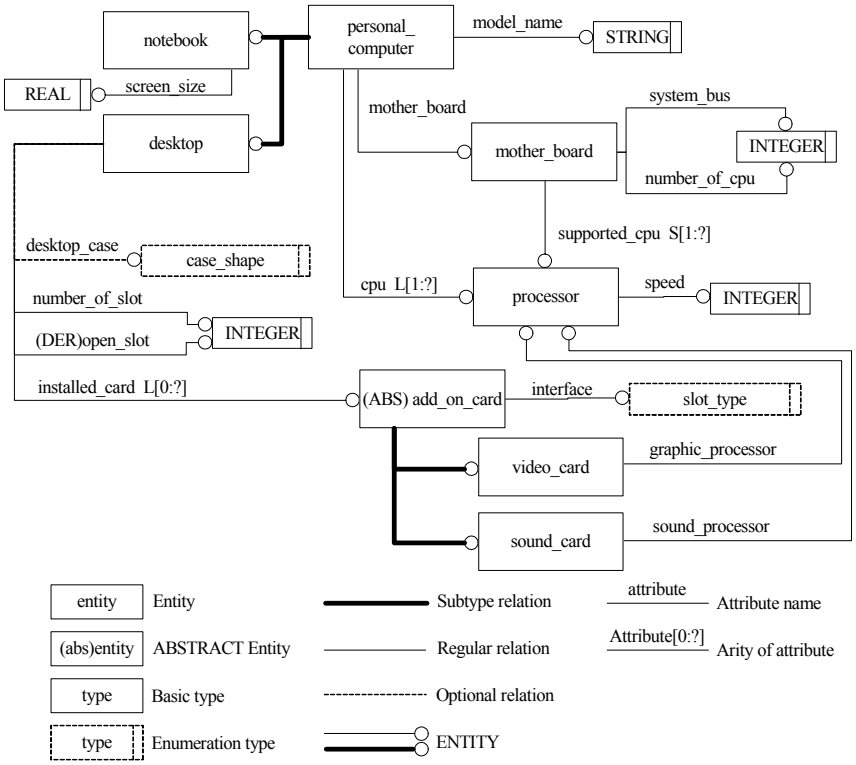

Figure 2: Product model definition of personal computer in EXPRESS-G

First query

Query for an entity itself returns multiple instances if it is not specified. First query should filter out subtype instance of the entity, because subtype shares the same table in mapping method (R1) in Figure 2. There are two types of

entity check in the first query. First one is checking subtype of the target entity to filter out subtype instance. Second one is checking supertype of the entity and the sibling entity. Second is ANDOR checking. ANDOR entity type can be ANDOR relation with supertype or the sibling of the entity. Once the subtype instance and the ANDOR entity instance are filtered out, attributes of the target entity are extracted from the database.

Query result is mapped to EXPRESS instance based on the GTCIS2SQL translation rules. An instance ID is assigned to the instance data and saved in memory when an instance is created according to the retrieved data. Instances in an instance model are interrelated. Saved instance ID is referenced in referencing instance as interrelated instances. First query also checks instance ID when creating new instance, because the instance can be created by other instance by referencing. Instantiating an instance requires instantiation of referenced instances prior to its creation.

Subsequent query

Value for an entity type attribute is a primary key of a referenced entity (which is translated into tuples). The primary key stored in the GTCIS2SQL database is assigned by the system when data is stored. This key value is not coming from the entity ID in a STEP Part 21 file. The primary task for the subsequent query is to figure out the exact type of the referenced entity, since the entity can be either one of the subtype including itself or even an ANDOR subtype combination. The exact type and its subordinates can only be obtained from the data dictionary. Queries are generated to scan each subtype tables; as a result, the number of query commands generated is relatively large. In Figure 2, such type checking for ‘installed_card’ of ‘desktop’ requires 2 queries. If ‘video_card’ or ‘sound_card’ is populated and has subtypes, new queries for subtypes are also required.

In a STEP Part 21 file, when an ANDOR subtype instances are referenced by another instance, the type name of the combined entities are also stored within the referencing instance. When an entity instance has the attribute whose type is SELECT, exact name of the selected type is required to convert the queried tuples into a valid STEP instance (e.g. an instance in a Part 21 file). GTCIS2SQL database stores such type information associated with relevant attributes to eliminate such subsequent queries. Once its type information and the primary key of the entity is retrieved, a query result can be converted into corresponding STEP instance.

SUMMARY

This paper provides a robust mapping method, instance population, and instance extraction for EXPRESS language product models implemented as

on a relational database. Distinctive improvement against previous approaches includes, 1) multiple inheritance, 2) ANDOR entity type, 3) select type, 4) aggregation attribute type, 5) data population for non-leaf entity, and 6) Part21 file extraction.

The future of GTCIS2SQL is to serve as a central data repository of CIS2 based product model data. It is aiming to support exchange data between heterogeneous applications, e.g. design, analysis, detailing and manufacturing applications. Now we have implemented robust relational database mapping, data population and data extraction, next step includes data merging, version control in detailed level (instance level), change propagation and change management.

References

Adachi, Yoshinobu: 2002, Overview of IFC model server framework, ECPPM2002

Ambler, S,,. The design of a robust persistency layer for relational databases. White paper, AmbySoft Inc., May 1998.

Crowley, A., and Watson, A., (2000) CIMsteel Integration Standards, Release Two, Volume 2, Implementation Guide, Steel Construction Institute and Leeds University.

Crowley, A., and Watson, A., (2000) CIMsteel Integration Standards, Release Two, Volume 4, Logical Product Model, Steel Construction Institute and Leeds University.

Crowley, A., Smith, A., and Watson, A., (2000) CIMsteel Integration Standards, Release Two, Volume 5, Conformance Requirements, Steel Construction Institute and Leeds University.

ECPPM 2002 eWork and eBusiness in Architecture, Engineering and Construction367-372

Fahl, Gustav and Risch, Tore: 1997, Query processing over object views of relational data, The VLDB Journal, 6, pp261-281

ISO DIS10303 Part 11, 1991 EXPRESS Language Reference Manual: External Representation of Product Definition Data, Ed. D Schenck, ISO TC184/SC4/WG5, Document N14

ISO TC184/SC4/, 1995, ISO 10303 Part 042, Draft ISO TC184/SC4/WG12

Loffredo, David: 1998, Efficient Database Implementation of EXPRESS Information Models

Morris, Katherine C.: 1990, Translating Express to SQL: A User's Guide, NISTIR 90-4341, NIST

Schenk D A, Wilson P R, 1994, Information Modeling the EXPRESS Way (Oxford University Press, New York)

Snyder, J.,Aygen, Z.,Flemming, U. and Tsai, J. (1995) “Sprout- A Modeling Language for SEED” Special Issue on Computers in Building Design, Journal of Architectural Engineering 1 (4)

Snyder, J.,Aygen, Z.,Flemming, U. and Tsai, J. (1995) “Sprout- A Modeling Language for SEED” Special Issue on Computers in Building Design, Journal of Architectural Engineering 1 (4)