

Modeling Product Ontology with CQL

H.Murayama , O.Lemarchand, & Y.Mizoguchi

Corporate Research & Development Center

Toshiba Corporation, Japan

ABSTRACT: For the selection of an industrial product, detailed product specification is indispensable. ISO13584 Parts Library (PLIB) is an International Standard that defines how the specification of a product can be exchanged using ISO10303 physical file format. However, this does not provide a standardized means of query about objects in the library, nor the differential transfer of ontology of products between different implementations based on the standard. Moreover, a standard gradually evolves over time in its syntax and semantics. Thus metadata migration as well as data migration between data models conforming to different versions of the standard becomes indispensable. This poses a serious problem since the entities in the PLIB are expressed in an object-oriented manner, while the database query languages currently available such as SQL (Structured Query Language) or OQL (Object Query Language) are either about relational databases that have no class hierarchy, or about Object-oriented databases whose class construction and queries depend on embedded language such as C++ or Java. CQL (Class Query Language), presented herein, is to address this problem. The language is a declarative query language about an ontology represented in an object-oriented class hierarchy, based on the mathematical class concept known as ZFC (Zermelo-Fraenkel set theory with axiom of Choice). Furthermore, the language enables Boolean operations between classes whose properties are not necessarily the same. Thus, the language is a theoretical extension of the SQL to the object-oriented modeling domain. This paper describes the mathematical foundation of CQL and illustrates the syntactic characteristics of the language.

1 INTRODUCTION

For the selection of an industrial product, detailed product specification is indispensable. ISO13584 Parts Library (often abbreviated as PLIB) is an International Standard (Pierra 1997,2001) that stipulates how the specification of a product can be exchanged using ISO10303 physical file format (so called STEP file or Part21 file format). However, this neither provides a standardized means of query about objects in the library, nor the standardized means of construction of class hierarchy within a PLIB-conformant database. Moreover, a standard does not stand still. It gradually evolves over time both in its syntax and its semantics. Thus, a differential transfer of ontology of products between different implementations, i.e., metadata migration as well as data migration between data models conforming to different versions of the standard becomes indispensable. This poses a serious problem since the entities in PLIB are expressed in an object-oriented manner, while the query languages currently available such as SQL (Structured Query Language) or OQL (Object

Query Language) (Cattell & Barry 1999) are either about relational databases whose tables themselves have no class hierarchy, or about Object-oriented databases whose class construction and queries are procedural and embedded in a programming language such as C++ or Java, thus largely dependent upon specific algorithms and implementations. CQL (Class Query Language) first introduced in (Mizoguchi, Murayama, Minamino 2002), is a solution to the problem. The language is a declarative query language about ontology represented in an object-oriented class hierarchy, based on mathematical class and set notion known as ZFC (Zermelo-Fraenkel-set theory with axiom of Choice). Furthermore, this language enables set-theoretic Boolean operations between classes of which the sets of properties and their respective orders are not necessarily the same. An early version of the language is already used for the reduction of client-server communication in "OmniPhaseTM", the world-first commercial implementation of web-oriented PLIB-LMS (Library Management System). The following part of this paper describes the mathematical foundation of CQL and illustrates syntactic characteristics of the language.

$$A' = \{x \mid x \in X, \varphi_1(x) \wedge \varphi_2(x) \wedge \dots \wedge \varphi_n(x)\} \quad (2.4)$$

2 MATHEMATICAL FOUNDATION

The notion of class employed in this paper, and henceforth in CQL specification, corresponds to the class notion in mathematical logic (Church 1956). Class and set differs in that the latter is assured to have members, while the former only states the logical conditions by set-theoretic logical formulae called proposition or properties, needed to be satisfied by the members of the class. However if there really exists a member that satisfies the condition or not is out of the consideration. Thus, a statement,

$$C = \{X \mid X \notin X\} \quad (2.1)$$

is a legitimate expression of a class but not of a set. However, except for such a pathological case known as Russell's paradox, it is safe to say that every set is simultaneously a class.

The notion of a class whose members are qualified by having the same set of properties can be expressed by a predefined set of logical formulae, known as Zermelo-Fraenkel set-theory with axiom of choice (ZFC)(Jech 1997), but does not necessarily coincide with the popular class notion in Object-oriented Programming language (OOP). This latter merely implies such an entity be a named aggregation of attributes of various data types, which allows inheritance of member attributes to its sub-entities, often being equipped with selective data encapsulation. Such a class in OOP neither guarantees that every class is a member of another class, nor the collection of those classes form an ACGS(Acyclic Graph Structure). In contrast, every class in CQL, or in mathematical logic, is a subset of *Universal Class*, or *Universe*, thus having the same root node. This class definition neither corresponds to that of semantic ontology in which the analysis of the meaning of vocabulary plays a significant role in the configuration of class hierarchy. In CQL, a subclass A' of class A can be defined strictly by the formula in the following;

$$A' \subseteq A \leftrightarrow \forall x(x \in A' \rightarrow x \in A). \quad (2.2)$$

Hence, the subclass A' must have all the properties that class A has and may have some additional properties. We may rewrite the above using a conjunction of several properties $\varphi_i(x)$ called "conjunctive normal form(CNF)", with respect to x of A , as in the following,

$$A = \{x \mid x \in X, \varphi_1(x) \wedge \varphi_2(x) \wedge \dots \wedge \varphi_m(x)\} \quad (2.3)$$

in the same manner, we may rewrite A' as,

then the condition (2.2) stands whenever $m \leq n$.

Apart from the special cases given in (2.1), all the Boolean set operations that apply to sets apply to classes as well. Exactly as ZFC describes for mathematical classes (Jech 1997), definitions,

$$A \cap B = \{x \mid x \in A \wedge x \in B\}, \quad (2.5)$$

$$A \cup B = \{x \mid x \in A \vee x \in B\}, \quad (2.6)$$

$$A - B = \{x \mid x \in A \wedge x \notin B\}. \quad (2.7)$$

stand for the classes in CQL.

Note that class A and class B do not necessarily have the same set of properties, nor those properties must be arranged in the same order in the two operand classes, provided they are uniquely identifiable, in clear contrast to the relational database theory founded by E.F.Codd, where only rows (i.e., x) are presumed to have no specific order, as long as they are uniquely identifiable(Codd 1970).

In this paper, an object called *class* means a mathematical class with the above definition and implies that an assembly of those classes forms a tree, and it must be distinguished from a table in Relational Database which has no class hierarchy, or in other words, which has only flat hierarchy. In the latter, the Boolean set operations are essentially applied to the instances of a table, or between tables that have the same set of properties(attributes), and the order of them (Codd 1970, Revesz 2002).

In addition to the Boolean set operations in the above, Cartesian product of sets with constraints, usually known by the name of "*cross join*" operation is also provided in CQL. This is formalized as in the following;

$$A \times B = \{(x, y) \mid x \in A \wedge y \in B\}. \quad (2.8)$$

where (x, y) signifies pairing of the two elements.

Note that the union of classes defined in (2.6) is the union of the elements of the two classes, while the join of two classes is a pairing of elements of the two sets selecting one element from each operand class. Combining the above with the member (instance / row) selection operations and property (column) selection (usually called "projection") operations, often symbolized as π and θ respectively, various forms of join operations such as equi-join, inner-join, and natural joins are expressed in the form;

$$A \circ B = \pi(\theta(A \times B)), \quad (2.9)$$

where \circ stands for any one of such join operations.

The CQL differs from OQL(Object Query Language) proposed by ODMG (Object Data Management Group) (Cattell and Barry 1999), in that CQL enables set-theoretic Boolean operations in opera-

tions to model a new class or to retrieve information from it, while in OQL, it uses methods defined in a class to do so. Thus in OQL, the actual creation of a class depends on how the constructor method is algorithmically defined as well as in which order the constructor arguments are defined.

In CQL, a class is defined by a set of properties but the order of declaration of those properties has nothing to do with class creation, nor with the data manipulation. In fact, each property and each class are identified by a unique code, exactly as in PLIB, named “property_BSU (Basic Semantic Unit)” and “class BSU” respectively. And the issuing organizations of those are globally identified by a code named “Supplier BSU”, whose uniqueness and integrity are guaranteed by the ICD (International Code Designator) defined in ISO6523(ISO/IEC 6523-1 1998, ISO/IEC 6523-2 1998). Furthermore, the collections of those properties and classes are bundled together as an exchangeable metadata, and given a special name, i.e., “dictionary” or “data dictionary”.

3 LANGUAGE FEATURES

CQL provides three tiers of definition language: a Class Modeling Language (CML), a Data Representation Language (DRL) and a Library Manipulation Language (LML). Unlike OQL which is a procedural language that provides only methods in the Object-oriented programming language sense, and SQL which can only manipulate relational tables and their content, CQL provides an efficient means to handle metadata (also called “dictionary” or “ontology”) as well as content data(or so called “instances”).

	CQL	SQL	OQL
Product Ontology	CML (create class alter class i.e.)	N.A.	Method
Database Schema	DRL (create, alter i.e.)	DDL (create, alter)	Method
Content Data	LML (insert, update i.e.)	DML (insert , update i.e.)	Method

Table 1: Comparison of Query Languages

A newer version of SQL was officially introduced as SQL-99 (once called “SQL3”), but it does not change the whole picture. In SQL-99 (P. O’Neil and E.O’Neil 2001, Gardarin 1999), a subtable is definable under existing one inheriting the attributes of a parent table using the following statement:

$$\text{create table } A \text{ of type } B \text{ under } C; \quad (3.1)$$

But this does not mean, subclass data access is uniformly expressed by a set-theoretic Boolean operations as we propose in this paper. In other words, in SQL-99, the scope of Boolean operation is still restricted to homogeneous relations. Theoretically speaking, the approach introduced as “Object Algebra” for object-oriented data models in (R.Alhadj and M. Arkun) shares some common ground with CQL, however, its definitions of Boolean operations are not based on ZFC and its query syntax is largely different from those of SQL and CQL, thus it cannot be compatible with SQL.

3.1 Data manipulation

Ancestral search and subclass search

First of all, CQL allows Ancestral Search. In other words, when processing a query, the search scope can be modified in order to reach a whole branch of the hierarchy, or only the subclasses or superclasses of a given class.

$$\text{select } * \text{ from } A; \quad (3.2)$$

$$\text{select } * \text{ from } A*; \quad (3.3)$$

$$\text{select } A.* \text{ from } A*; \quad (3.4)$$

$$\text{select } * \text{ from } A\%; \quad (3.5)$$

$$\text{select } * \text{ from } A!; \quad (3.6)$$

In the query (3.2), the search scope is limited to the class A only. In the query (3.3), the “*” operator extends the scope of (3.2) to include all the subclasses of A. If “A.*” are used as in (3.4) instead of “*”, this means, only the properties that are applicable at the level of A, but not the properties that are defined in any of its subclasses shall be searched for their values. Finally, the queries (3.5) and (3.6) extend the scope of (3.2) respectively to include all the superclasses(in general, a class may have several parents) of A, or to all the direct ancestors(for identification purpose, only one of the several parent classes is defined as the ancestor) of A only.

Boolean operations

Boolean set-theoretic operations between operand classes modify the search scope of the query in the class hierarchy. The expressions are given as below.

$$\text{select } * \text{ from } A* \& B*; \quad (3.7)$$

$$\text{select } * \text{ from } A* + B*; \quad (3.8)$$

$$\text{select } * \text{ from } A* - B*; \quad (3.9)$$

These three expressions (3.7), (3.8), and (3.9) correspond respectively to the mathematical expressions of class intersection, class union and class subtraction, defined in (2.5), (2.6), (2.7) respectively. In place of ‘&’ for the symbol of intersection operation,

a literal “AND” is also allowed to give more readability.

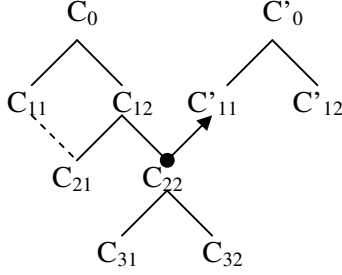


Figure 1: Sample Class Tree

For example, in a class hierarchy like the one depicted above in Figure 1, if the **from** clause is written “**from C₁₂**”, the search scope is strictly limited to the class **C₁₂** itself. But if the **from** clause is “**from C₁₂***”, the search scope is the class **C₁₂** and all its subclasses. In the case of a **from** clause like “**from C₀* & C₁₁***”, the search scope is then restricted to **C₂₂** and all its subclasses. Note that these subclasses represent conjunction, or “AND”, of the two classes **C₀*** & **C₁₁*** having the properties of the classes, by virtue of inheritance.

Likewise, a Boolean subtraction operation, “**from C₀* - C₂₂***” means that the class **C₂₂** and its subclasses are removed from the search scope of **C₀***.

In the figure 1, an arrow points from **C₂₂** to **C₁₁***, signifying a special type of multiple inheritance, named “partial inheritance”, that may or may not inherit all the properties of the class **C₁₁*** pointed by the arrow, thus being distinguished from the true parent class **C₁₂** used for identification purpose of **C₂₂** in the hierarchy.

Join Operations

A typical syntax of the join operations corresponding to (2.8) in CQL may be shown as in the following;

$$\text{select * from A, B;} \quad (3.10)$$

The more general syntax of the join operations in CQL may be summarized removing inessential language options as in the following structure;

$$\begin{aligned} &\text{select } \langle \text{property_list} \rangle \text{ from } \langle \text{class_list} \rangle \\ &[\text{where } \langle \text{property_id} \rangle \\ &\quad \text{OPERATOR } \langle \text{value_expression} \rangle \\ &\quad \{ [\text{AND} \mid \text{OR}] \\ &\quad \langle \text{property_id} \rangle \\ &\quad \text{OPERATOR } \langle \text{value_expression} \rangle \\ &\quad \}^* \\ &]; \end{aligned} \quad (3.11)$$

In the above constructs, **OPERATOR** may be one of the following operators; ‘<’, ‘>’, ‘=’, ‘<=’, ‘>=’, ‘<>’, ‘LIKE’ .

Suppose that in Figure 1, **C₁₂** has properties **p1** and **p2**, and **C₁₁*** has the properties **p2**, **p3**. Then, some of the examples of using the above construct may be as in the following;

$$\begin{aligned} &\text{select } p1, C_{12}.p2, C'_{11}.p2, p3 \text{ from } C_{12}, C'_{11} \\ &\text{where } C_{12}.P2 = C'_{11}.P2; \end{aligned} \quad (3.12)$$

or using ‘*’ for including the subclasses of **C₁₂**, the above may be rewritten:

$$\begin{aligned} &\text{select } p1, C_{12}.p2, C'_{11}.p2, p3 \text{ from } C_{12}^*, C'_{11} \\ &\text{where } C_{12}.P2 = C'_{11}.P2; \end{aligned} \quad (3.13)$$

Note that second example in the above applies the join operation to **C₁₂** and all its subclasses, however, these subclasses are viewed at the level of **C₁₂**, therefore, in the property list “**p1, C₁₂.p2, C₁₁.p2, p3**”, no property particular to any subclass of **C₁₂** may be specified. However, when ‘*’ is used as in the following for the property list, the search will list the values of all the properties found in the hierarchical search path;

$$\text{select* from } C_{12}^*, C'_{11} \text{ where } C_{12}.P2 = C'_{11}.P2; \quad (3.14)$$

To limit the list of properties for projection operation to the level of **C₁₂** and **C₁₁***, it is necessary to write instead, as in the following;

$$\begin{aligned} &\text{select } C_{12}.*, C'_{11}.* \text{ from } C_{12}^*, C'_{11} \\ &\text{where } C_{12}.P2 = C'_{11}.P2; \end{aligned} \quad (3.14)$$

It is necessary to note about the constraints to domain of properties that may appear on the property list; except for the properties possessed by or inherited into the classes specified in **from** clause, no subclass properties other than * may appear in the property list of the **select** construct.

3.2 Metadata manipulation

From the CQL point of view, querying metadata is nothing different from querying content data; for the data as well as meta-data are accommodated in the similar structures called *table* belonging to a class. Therefore, apart from the create/alter/drop/update commands of the CML, the retrieval of meta information is performed by a select statement whose search scope is entitled to any kind of ontological entity:

- Project
- Dictionary
- Class
- Applicable or visible property
- User

- Extent
- Table

For example, the following select syntax,

```
select class from DI
  where super_class = C12; (3.15)
```

retrieves all the classes in the dictionary *DI* whose *super_class* is *C₁₂*. As you clearly see this must return *C₂₁* and *C₂₂* in the example of class hierarchy given in **Figure 1**. Of course, here *DI*, and *C₂₁* and *C₂₂* are represented by their *class_BSUs*. As you see in the next section, in CQL, a dictionary is in fact a special type of class, called partially ordered collection whose element has relations with other classes and updates the property values of those related.

Likewise, the following construct gives all the applicable properties of class *C₁₂*

```
select applicable_properties from C12; (3.16)
```

and the next example gives all the *super_classes* of the ancestors of *C₁₂*,

```
select super_class from C12%; (3.17)
```

This approach of meta data modeling and query has an advantage of allowing any ontology following the object-oriented pattern to be modeled and modified with simple query syntax.

3.3 Collections, or model modelers

The modeling of aggregates, or collections as referred hereafter, is in CQL a very high level data description and abstraction functionality, that is deep-rooted in fundamentals of mathematics, i.e., set and class theory. This in fact serves as a basic generator of meta-data as well as data.

First of all, due to the data typing mechanism of CQL, which distinguishes primitive data types (integer, real, string, etc.) from user defined data types, there are two main categories of collections. On one hand, homogeneous collections whose elements are all of the same type, and on the other hand heterogeneous collections whose elements can be of any type, primitive as well as user defined. Therefore, a collection can gather for example integer elements and class-instances at the same time. Such a distinction – heterogeneous and homogeneous – is necessary since there is no common super-type between primitive data types and user-defined data types (the equivalent of an Object type in common object oriented programming languages). From this point of view, a collection is nothing but a gathering of elements.

Any collection in CQL can be described with a minimal and a maximal cardinality, with a condition on the uniqueness of its elements.

Besides, CQL provides different specializations of the basic concept of collection.

```
{“butterfly”, 43, ‘u’,TOPAS/DEMO040} (3.18)
```

```
{1,3,16,8,43,27} (3.19)
```

where

(3.18) is an heterogeneous collection, and

(3.19) is an homogeneous collection of integers.

In the same manner, the declaration;

```
Collection of 6 int_type (3.20)
```

designates an homogeneous collection of integers, with a capacity of 6 elements.

Indexed collection

An indexed collection embodies the concept of dictionary. Each element of the index allows the access to one element of the collection. Such a collection must satisfy the following conditions to ensure the coherence of the index:

- Each element of the index is unique.
- Each element of the index is associated with one and only one element of the collection.

CQL does not impose any restriction on the type of the index. In other words, an indexed collection can be seen as two collections related to each other by their elements.

```
“Collection of 3 string_type indexed with int_type” (3.21)
```

designates a homogeneous collection of 3 strings, indexed by integers.

```
“Collection indexed by (‘name’ string_type, ‘age’ int_type, ‘address’ string_type)” (3.22)
```

stands for a collection with three elements, indexed respectively by ‘name’, ‘age’ and ‘address’. A specific data type can be attributed to each element.

Partially ordered collection

A partially ordered collection is a collection associated with a binary relation called partial-order. A partial order is a binary relation that is reflexive, transitive and antisymmetric (Gries & Schneider 1993).

This kind of collections is aimed at describing structures where any couple of elements can be related. Among others, graphs and trees are partially ordered collections. In the case of a graph, the partial order relation could be “is accessible from”, and in

the case of a tree, we could choose the relation “is an ancestor of” assuming that each element is one of its ancestors.

Ordered collection

An ordered collection is a natural extension of a partially ordered collection. In an ordered collection (or totally ordered collection) any two elements are comparable. Different types of orders can be distinguished:

- The collection is implicitly ordered when no specific order is provided. In this case, the elements are ordered implicitly: the first element added is the first element of the collection, etc.

$$\text{ordered collection of } 7 \quad (3.23)$$

- Any homogeneous collection of primitive types can be ordered by value, the natural order relation is then chosen. (“less than” for numerical data types, lexicographic order for strings and characters)

$$\text{collection of } 10 \text{ int_type ordered by value} \quad (3.24)$$

- If the ordered collection is also an indexed collection, it can be ordered either by value or by index.

$$\begin{aligned} &\text{collection of } 5 \text{ string_type indexed with real_type} \\ &\text{ordered by index} \end{aligned} \quad (3.25)$$

Cyclic collection

A cyclic collection is also an extension of an ordered collection. Aggregate data types such as SET, LIST are planned to be introduced for SQL-4, in addition to ARRAY that already adopted in SQL-99(P. O’Neil and E.O’Neil 2001), however, so long as we investigated, a cyclic collection is first to appear in query language references. In the case of a cyclic collection, the index takes its value in $\mathbb{Z}/n\mathbb{Z}$ where n is either the maximum size of the collection or its current capacity when no maximum size was provided. This represents the set of the equivalence classes for the relation “modulo n”. Thus, (a, b, c) and (b, c, a) are identical (with one shift) while (a, b, c) and (a, c, b) are not.

The concept behind cyclic collection is to describe a structure with periodical occurrences of data. For example, it may be useful to recognize a

periodical signal or pattern data only from a selection of extracted samples.

As intended to be a superset of SQL, CQL includes support for usual SQL commands.

Relational tables

Relational tables can be created and managed within the CQL language.

In the CQL model, every object whose modification has direct effect on the behavior or configuration of the library is considered a Resource. Otherwise, it is either part of the library or the classes of user that controls the user access to each class in library and resource. As a result, relational tables, as any object, can belong either to the library or the resources.

In order to create, alter, or drop a relational table, the user should use the following syntaxes:

$$\text{create table } \langle \text{table_name} \rangle \langle \text{field_list} \rangle, \langle \text{constraint_list} \rangle; \quad (3.26)$$

$$\text{alter table } \langle \text{table_name} \rangle [\text{ADD}|\text{MODIFY}|\text{DROP}] \langle \text{field_list} \rangle; \quad (3.27)$$

$$\text{drop table } \langle \text{table_name} \rangle; \quad (3.28)$$

The only constraint for those relational tables is to use CQL datatypes instead of SQL datatypes. This constraint allows not only to have a uniform datatype model, but also to work on both structures simultaneously within the same environment (object-oriented structure and relational structure).

For instance, it is relevant to process data from the class hierarchy and from a relational table via a join operation.

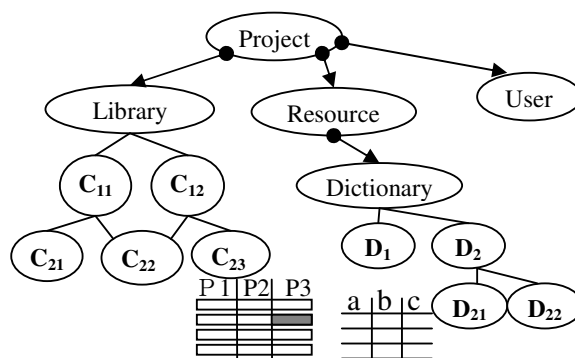


Figure 2: Resources and Library in CQL

In the example shown in Fig. 2, it is possible to join the table T1 with the class C23 for instance:

$$\text{select table.T1.a, C23.P2 from table.t1,C23*}; \quad (3.29)$$

This command would realize the join between the table D1 and the content associated to the class C23

and select only one property of the class and one field of the table.

4 ONTOLOGY AND ITS TRANSFER

4.1 Table format

Ontology in the CQL sense consists basically in a set of classes, each class having intrinsic attributes, such as preferred names in different languages, definitions, synonyms, a superclass, a universal identifier (BSU code), a set of visible properties and a set of applicable properties. The names of the attributes are in fact BSUs, and allow an unambiguous reference.

Obviously the classes of a dictionary can be described in a table (in the relational sense) without ambiguity. As reminded above, all classes are characterized by the same attributes(properties) , and there is a bijection between these attributes and the fields of a relational table. Besides, since all these attributes are identified by a basic semantic unit, their order of appearance in the table is irrelevant.

class_BSU	preferred_name.en	...	applicable	visible
AAA	motorized vehicle	...	(P ₁ ,P ₂)	()
:	:	:	:	:
NNN	car	...	(P _{n-1} ,P _n)	(P _k ,...,P _p)

Table 2: storing classes in a table

With the same approach, the properties used in a dictionary can also be described in a table: a property consists in preferred names in several languages, definitions, a universal identifier, a name_scope (a reference to the class where the property is defined), a value format, a unit and a data type.

As well as classes or properties, data types are complex entities which also have intrinsic properties of their own: a universal identifier (data_type_BSU), possibly a value format, etc. In order to describe exhaustively an ontology, three tables are needed.

Nevertheless the Boolean algebra allows us to use only two tables instead of three when all the data types of the ontology are explicitly used. A join operation is indeed possible between the table of properties and the table of data types, provided each attribute is essentially referred using special identifiers, i.e., BSUs.

Consequently, the two representations are equivalent, and a dictionary can be described in an object-oriented manner as well as in a relational table manner. This equivalence ensures the validity of the translation of one format into the other, and also insures the reversibility of such a conversion.

4.2 Transfer of an ontology with the table format

As a result of the equivalence between an object-oriented model and its “table format” representation, transferring a whole ontology can be reduced to transferring its table representation with a simple CSV(Comma Separated Values, used in many spreadsheet applications) file.

CQL embeds commands to create and alter dictionaries out of CSV files and conversely.

From table to dictionary

In the following expression, <entity_type> stands for one of class, property or data_type, and <source_type> stands for CSV or table. Indeed, CQL handles tables as files or as actual relational tables stored within the database.

```
<create_command> ::= create dictionary <dic_id> with
  <entity_type> (<CQL attributes>)
  {, <entity_type> (<CQL attributes>)}
  as select <attributes> from
  <table_id> of <entity_type> in <source_type>
  {, select <attributes> from
  <table_id> of <entity_type> in <source_type>} (5.2)
```

```
<alter_command> ::= create dictionary <dic_id>
  as select <attributes> from
  <table_id> of <entity_type> in <source_type>
  {, select <attributes> from
  <table_id> of <entity_type> in <source_type>} (4.1)
```

From dictionary to table

The reverse operation consists in creating tables – either within the database or in CSV format – from a dictionary.

```
create <table_identifier> as <entity_type> [in CSV]
  from dictionary <dictionary_identifier>
  [where <condition>] (4.2)
```

The “where” clause is optional but allows to apply filters on the entities one wants to export in a table.

4.3 Parceling format

The parceling format or parceling is XML based. The concept of parceling is to encapsulate data within XML tags as a payload just like in the SOAP protocol, but in particular for a CQL/SQL table format.

XML is a widespread standard for exchanging data over the Internet. XML has advantages, espe-

cially with respect to its ability to describe a hierarchical structure such as a dictionary, although its actual capacity to check the consistency of data has a long way to come. Besides, an XML file is totally platform independent and may bypass the problem of multiple character-set encoding.

The parceling takes advantage of these features and provides CQL with a useful medium to exchange information, in small parts as well as a whole dictionary. With a table format, several files are needed to perform a complete transfer of ontology, but with the Parcel format, anything can be described through the use of different XML tags. An example of such a parceling format is as follows;

```
<?xml version="1.0" encoding="Shift_JIS"?>
<dictionary dic="AAA" supplier_bsu_code="DDD">
  <dic_element>
    <class>
      #BSU,ID,preferred_name,super_class
      ,AAA_ROOT,Schema root,Universal
      ,AAA_C01,Car,AAA_ROOT
      ,AAA_C02,Sedan,AAA_C01
    </class>
    <property>
      ...
    </property>
  </dic_element>
</dictionary>                                     (4.3)
```

5 FUTURE PERSPECTIVE

In this paper, a new database language short named "CQL" is presented. This language has an advantage over SQL in that it can model object-oriented class hierarchy. It also has an advantage over OQL in that it is a declarative query language and may employ Boolean set-theoretic operations for data manipulation and metadata manipulation. This gives ease to end-users and eliminates the necessity to know about the detail of the class definition or the specification of embedding language. The future developments of CQL are now targeted at modeling not only a static ontology but also the behavior of each class, function of its environment. The foundation of the behavioral modeling in the object-oriented world is based on message passing between classes. When a class receives a message, it chooses itself the way it reacts.

With the implementation of CQL in our laboratory, construction of a commercial PLIB-LMS (Library Management System) named "OmniPhase" has completed its first phase. In the system, the CQL is exploited for two objectives; one is to facilitate the communication between LMS and its client programs, the other is to provide an API or a user interface to external users or programs. The system is

mostly programmed in JAVA™ language, and the server employs servlet technology and most of the client side programs are realised with applets including swing libraries. The interface to external program is currently programmed in JAVA™ or in C++. This interface allows a connection to an external program with an execClassQuery statement, like the one available in JDBC (Java Database connection).

Currently, an advanced version of CQL based PLIB-LMS is under construction in our laboratory. The new system sits on the top of PostgreSQL database system, and almost all the data and metadata manipulation of the new system is through CQL.

More detailed specification of CQL will be available from our PLIB web site specified in the following URL:

<http://www.toplib.com/en/cql.html>

REFERENCES

- Jech, T. 1997. "Set Theory" Springer-Verlag
- Church, A., 1956, "Introduction to Mathematical Logic", Princeton University Press.
- Codd, F., 1970, "A Relational Model of Data for Large Shared data Banks", Communications of ACM, Vol.13, No 6, 1970
- Cattell, R.G.G. & Barry, D. K, 1999, "The Object Data Standard: ODMG 3.0. " Morgan Kaufmann Publishers.
- ISO/IEC 6523-1, 1998, "Identification of organization identification schemes", ISO/IEC
- ISO/IEC 6523-2, 1998, "Registration of organization identification schemes", ISO/IEC
- Pierra, G.1997. "Description methodology: Methodology for structuring parts families", ISO (reference is available from www.plib.ensma.fr)
- Pierra G. 2001. "Logical resource: Logical model of supplier library. " ISO (reference is available from www.plib.ensma.fr)
- Mizoguchi, Y., Murayama, H & Minamino, N. 2002, "Class Query Language and its application to ISO13584 Parts Library Standard", ECEC2002
- O'Neil.P. & O'Neil, E. 2001, "Database: Principles, Programming, and Performance" Academic Press
- Alhadj, R., Arkun, M. 1992, "Queries in Object-Oriented Database Systems", CIKM92, Springer-Verlag.
- .Revesz, P., 2002. "Introduction to Constraint Databases", Springer-Verlag
- Gries ,D. and Schneider, F., 1993 "A Logical Approach to Discrete Math", Springer-Verlag
- Gardarin, G., 1999, "Bases de Données Objet et Relationnel", Eyrolles