MAPPING OF OWL ONTOLOGY CONCEPTS TO RDB SCHEMAS

Ernestas Vysniauskas, Lina Nemuraite
Kaunas University of Technology, Department of Information Systems, Studentu 50-313a, Kaunas, Lithuania, Ernestas.Vysniauskas@stud.ktu.lt, Lina.Nemuraite@ktu.lt

Abstract. Modern technologies of Semantic Web, the growing complexity of information systems, the needs of knowledge bases and smart Web agents require conceptual models to gain an improved form of semantic knowledge models – i.e. ontology. Currently, the main technique of storing ontology is based on files containing descriptions of ontology in RDF/RDFS or OWL. On the other hand, the relational database technology ensures the best facilities for storing, searching, updating and manipulating information of the problem domain. The purpose of this paper is to propose the principles of mapping OWL concepts to relational database schemas, along with algorithms for transformation of OWL ontology descriptions to the relational database.

Keywords. Ontology, relational database, OWL, mapping, transformation.

1 Introduction

Ontology descriptions are typically used in Semantic Web/Web2.0, but nowadays they find more and more adaptability in everyday Information Systems. Also, modern Information Systems tend to incorporate various forms of knowledge, e.g. business rules [10, 13] that are closely related with ontologies. Well-formed ontology must have correct syntax and unambiguous machine-understandable interpretation, so it is capable to clearly defining fundamental concepts and relationships of the problem domain. Ontologies are increasingly used in many applications: business process and information integration, search and navigation that require scalability and performance, efficient storage and manipulation of large scale ontological data. In such circumstances, storing ontologies in relational databases are becoming the relevant needs for Semantic Web and enterprises.

For ontology development, Semantic Web languages are dedicated: Resource Description Framework (RDF) and schema RDFS, and Web Ontology Language (OWL) that consists of three sublanguages – OWL Lite, OWL Description Logic (DL) and OWL Full. When ontology based systems are growing in scope and volume, reasoners of expert systems are becoming unsuitable. J. Lee and R. Goodwin [15] propose the database-centric approach to ontology management support and notice that such methodology is still in its infancy.

Methodologies for transforming Entity-relationship and Object-oriented (often expressed in UML) conceptual models to relational database structures, transformations between relational and XML schemas, UML models and XML schemas are well-established and implemented in CASE tools. Similarly, Ontology Definition Metamodel, initiated by OMG [17], defines transformations between OWL, UML, ER and other modelling languages, where Simple Common Logic is chosen for definition of constraints. One possible way to relate ontological information described by ontological language with relational schemas is to use standard methodologies, e.g. generating UML models from standard ontology descriptions (e.g., OWL) and then relational schemas from UML models. Another way is a direct transformation from OWL to relational schema.

There are some proposals for transforming ontology to relational databases; however, these approaches (e.g. [1, 3, 8]) are mainly straightforward and still incomplete, or obtained relational structures are not applicable for real information systems. While criteria of completeness and performance require for some compromise, this research is directed towards completeness and semantics preservation, i.e. bidirectional reversible transformations between OWL ontology and databases.

We propose algorithm which fully automatically transforms ontologies, represented in OWL, to RDB schemas. Some concepts, e.g. ontology classes and properties are mapped to relational tables, relations and attributes, other (constraints) are stored like metadata in special tables. Using both direct mapping and metadata, it is possible to obtain appropriate relational structures and do not lose the ontological data.

The rest of the paper is organized as follows. Section 2 presents mappings between ontological and relational concepts. Section 3 is devoted to transformation from OWL ontology to relational database. Section 4 presents the implementation of the proposed approach. In section 5, the related work is considered and section 6 draws conclusions and highlights the future work.

2 OWL/RDB Concepts and their Mapping

In this section, we present our approach combining mappings of OWL concepts to RDB concepts and storing the problematic (in mapping sense) knowledge in metadata tables.
2.1 OWL Classes and RDB Tables

OWL classes (Fig. 1) provide an abstraction mechanism for grouping resources with similar characteristics [17]. Like RDFS classes, every OWL class is associated with a set of individuals (instances), called the class extension. A class has the intentional meaning (the underlying concept) which is related but not equal to its class extension. Thus, two classes may have the same class extension, but still be different classes. OWL classes are described through “class descriptions,” which can be combined into “class axioms”. A class description, as shown in Figure 1, describes the OWL class, either by a class name or by specifying the extension of an unnamed anonymous class. OWL distinguishes six types of class descriptions: class identifiers (URI references), exhaustive enumerations of individuals, property restrictions, intersections of class descriptions, unions, and complements.

The first type is special in the sense that it describes a class through a class name (syntactically represented as a URI reference). The other five types of class descriptions may describe anonymous classes but the naming is common for the convenience, reuse, and readability purposes.

For a relational database, we use the metamodel (Fig. 2), which comprises a part of Common Warehouse Metamodel (CWM) [5] that currently is under extension to more powerful CWM 2.x named as “Information Management Metamodel” (IMM) [12].

In the relational metamodel, a ColumnSet represents any form of relational data. A NamedColumnSet is a catalogued version of a ColumnSet, which is owned by the Schema. A NamedColumnSet can be a logical
View or a physical Table. Instead of being a NamedColumnSet, a ColumnSet can be a QueryColumnSet, which is the result of an SQL query. Columns are associated with an SQLDataType, using the type association from StructuralFeature to Classifier inherited from ObjectModel Core. Figure 2 shows the original two data types: simple type and distinct type. Simple types are defined by the SQL standard; however, some RDBMS implementations use additional types. An SQL distinct type is defined from a simple type.

When we are converting the OWL ontology description to relational database schema, we map one ontology class to one database table. As the name of an ontology class is unique in the whole ontology, and even instances of the ontology class have unique names, we propose to create a primary key for corresponding table automatically, by adding some suffix to the class name, e.g. "_id". This mapping and the example are shown in Figure 3.

![Figure 3. Mapping OWL class to RDB table](image)

When transforming ontology representation in OWL to relational database schema, we suggest creating one table for every class in ontology with one-to-one relations between classes and their subclasses. So the constructor "rdfs:subClassOf" in OWL is mapped to one-to-one relation in RDB (Fig. 4).

![Figure 4. Mapping OWL subclass constructor to RDB 1:1 relation](image)

### 2.2 OWL Properties, RDB Columns and Constraints

As shown in Figure 5, OWL refines the notion of the RDFS property to support two main categories of properties as well as annotation properties that may be useful for ontology documentation:

- **Object properties**, which relate individuals to other individuals;
- **Data type properties**, which relate individuals to data values;
- **Annotation properties**, which allow to annotate various constructs in ontology;
- **Ontology properties**, which allow saying things about ontology itself.

The distinction made between kinds of annotation properties (i.e., annotation vs. ontology properties) are needed to support OWL DL semantics. In addition, a number of property axioms is provided for property characterization.

A functional property is the property that can have only one (unique) value $y$ for each instance $x$, i.e. there cannot be two distinct values $y_1$ and $y_2$ such that the pairs $(x, y_1)$ and $(x, y_2)$ are both instances of this property. Both object properties and data type properties can be declared as “functional.” For this purpose, OWL defines the built-in class owl:FunctionalProperty as a special subclass of the RDFS class rdf:Property.

If a property is declared to be inverse-functional, then the object of the property statement uniquely determines the subject (some individual). More formally, if we state that $P$ is an owl:InverseFunctionalProperty,
then this asserts that a value $y$ can only be the value of $P$ for the single instance $x$, i.e., two distinct instances $x_1$ and $x_2$ cannot exist such that both pairs $(x_1, y)$ and $(x_2, y)$ are instances of $P$.

Data type properties are used to link individuals to data values. A data type property is defined as the instance of the built-in OWL class owl:DatatypeProperty. The object property relates the individual to other individuals. An object property is defined as the instance of the built-in OWL class owl:ObjectProperty.

In the Relational model, a key is a set of attributes that defines uniqueness among the instances of the class. The Relational model extends the UniqueKey class to UniqueConstraint (Fig. 6). Similarly, the Relational package uses KeyRelationship from the Foundation package as the basis for the ForeignKey. The generic associations of the Foundation - UniqueKey, KeyRelationship, Class and StructuralFeatures - are inherited by associations between UniqueConstraint, ForeignKey, Table, and Columns in the Relational package.

The object property in OWL ontology relates the individual to other individuals, and the Foreign Key associates columns from one table with columns of another table, so for converting the OWL ontology description to the RDB schema, we map the object property to the foreign key (Fig. 7). Depending on the local
cardinality of some class property, one-to-many or many-to-many relation between tables of classes are created. In a case of many-to-many relation, an intermediate table must be created.

**Figure 7. Mapping OWL object properties to foreign keys**

A data type property in OWL ontology relates an individual to a data value. We map data type properties to relational database columns of the tables corresponding to the domain classes of these properties. Because a range of a data type property is the XML schema data type, we map XML schema data types to corresponding SQL data types (Fig. 8).

**Figure 8. Mapping OWL data type property to column**

### 2.3 OWL Restrictions

In OWL, the class owl:Restriction is defined as a subclass of owl:Class (Fig. 9). A restriction class should have exactly one triple linking the restriction to the particular property, using the owl:onProperty. The restriction class should also have exactly one triple that represents the value of the cardinality constraint (e.g. exactly 1) on the property under consideration. Property restrictions can be applied both to data type properties and object properties.

**Figure 9. The OWL restrictions ([17])**

The cardinality constraint owl:cardinality is a built-in OWL property that links a restriction class to a data value belonging to the range of the XML Schema datatype xsd:nonNegativeInteger. A restriction containing an owl:cardinality constraint describes a class of all individuals that have exactly N semantically distinct values (individuals or data values) for the property concerned, where N is the value of the cardinality constraint. Syntactically, the cardinality constraint is represented as an RDFS property element with the corresponding rdf:datatype attribute. Similarly, the cardinality constraints owl:maxCardinality and owl:minCardinality are represented.

The AllValuesFromRestriction describes a class for which all values of the property under consideration are either members of the class extension of the class description or are data values within the specified data range. In other words, it defines a class of individuals x for which holds that if the pair (x, y) is an instance of P (the property concerned), then y should be an instance of the class description or a value in the data range,
respectively. The HasValueRestriction describes a class of all individuals for which the property concerned has at least one value semantically equal to V (it may have other values as well). The SomeValuesFromRestriction describes a class for which at least one value of the property under consideration is either a member of the class extension of the class description or a data value within the specified data range. It defines a class of individuals x for which there is at least one y (either an instance of the class description or the value in the data range) such that the pair (x, y) is an instance of P (the property concerned). This does not exclude that there are other instances (x, y') of P for which y' does not belong to the class description or data range.

When we are converting the OWL ontology description to the relational database schema we want to preserve all semantic information of ontology constraints. For this purpose we suggest to save this information in special metadata tables. Each type of restriction has its own metadata table with two columns “DomainClass” and “RangeClass”, where we save table names of converted classes. Metadata tables for AllValuesFrom and SomeValuesFrom restrictions also have column “RestrictionClass” which points to the table of the corresponding restriction resource class (Fig. 10). The HasValue restriction metadata table has the column “Value” for storing the value of the restricted resource of the corresponding property.

![Figure 10. Mapping OWL value restrictions to metadata tables](image)

Metadata table “Cardinality” has three additional nullable columns for each type of OWL cardinality restriction (Fig. 11). E.g. if we have a property with cardinality restriction owl:cardinality equal to “3”, after transformation this metadata table has value “3” in the field “Cardinality”, and others two types “MinCardinality” and “MaxCardinality” has value “NULL”.

![Figure 11. Mapping OWL cardinality restrictions to metadata tables](image)

3 Transformation of OWL Ontology to Relational Database

In this chapter we propose an approach of a transformation algorithm, which parses OWL documents and generates DDL scripts, containing database descriptions with included domain ontology constraints. The process of transforming ontology into relational database begins when designer initiates transformation of domain ontology described in OWL file into relational database. Transformation tool checks the correctness of OWL syntax and transforms the ontology to the relational database starting from ontology classes, then transforming object and data type properties, constraints and, finally, fills the database with instances of classes.

Algorithm, transforming ontology classes into relational database tables, uses breadth-first search algorithm that goes across one hierarchical level of ontology class tree, thus root classes are parsed first, then their subclasses, and so on (Fig. 12). The table in the relational database is created for every class in ontology with one-to-one relations between classes and their subclasses. Breadth-first search algorithm guarantees that when some subclass is being created, its parent class in hierarchy has already been created.

After creating ontology class tables, algorithm performs transformation of object properties into RDB relations between class tables also using breadth-first search. First, it parses properties that do not have properties of the higher hierarchical level, then their sub-properties, and so on. Depending on the local cardinality of the class property, one-to-many or many-to-many relation between tables of classes are created. In a case of many-to-many relation, an intermediate table is created, replacing one many-to-many relation with two one-to-many relations.
Next, data type-properties are transformed into RDB data columns. Algorithm searches and parses all data type properties in series. According to rdfs:domain value it finds database table and creates data column with the name of the property. Column field type is set according to the property rdfs:range value. Data type property transformation is finished when all data type properties are parsed.

After transformation of data type-properties into RDB data columns, ontology constraints are transformed into RDB metadata tables also by performing breadth-first search. First it parses constraints of root class properties, then their subclass constraints, and so on. If a class has constraints on the property, algorithm transforms them into metadata tables. All constraints of the particular class are parsed in series. Every type of constraint has its own table with the name of the constraint type.

At the last stage of transforming domain ontology into relational database, transformation tool inserts all instances of classes into the created database.

4 Implementation

According to the proposed algorithms, we implemented a prototype tool for transforming domain ontology described in OWL into relational database schema. The tool was implemented as a plug-in for a popular ontology editor Protégé. It consists of the user interface module, Protégé OWL plug-in for OWL ontology management, Jena API and the OWL to RDB transformation module (Fig. 13).

User interface module is used for helping a designer to choose a desired OWL file, represent ontology graphically, define the connection to the relational database server and execute other needed commands. Protégé OWL ontology management module performs validation of ontology description syntax and imports OWL file structures into Jena API objects. OWL to RDB transformation module examines ontology objects, performs OWL concept transformation to relational database schemas, forms and executes SQL queries, and implements the connection between transformation tool and database server using JDBC driver.
The graphical interface of the transformation tool is presented in Figure 14. The interface shows the hierarchy of ontology classes, the window for checking and editing the generated DDL script, and buttons for creating a database, generating DDL script and saving it in SQL file. The tool was tried and it worked well for transforming a set of experimental ontological descriptions.

5 Related Work

When discussing OWL to RDB transformation, we should notice that there are much more approaches for inverse mapping – i.e. from relational databases to ontology (the survey is given in [19]). One of the main problems of sharing data on the Semantic Web is the lack of semantically rich data, since most of information is stored in relational databases [4, 6, 7, 16]. The relational OWL query language is proposed for mapping legacy data stored in relational databases to the Semantic Web. The approach presented in [11] allows importing data sources into an RDF repository. The object domain model is mapped to ontology and to the persistence model. Specifying mappings requires model annotations performed by domain experts. All other processes are automated and data transformations are generated from the mappings via model weaving techniques.
Some authors research direct transformation, e.g. in [18], converting the selected data from a relational database to a RDF/OWL ontology document is based on templates. Another source [9] uses RDB2OWL mechanism in the process of achieving efficient interoperability of heterogeneous information systems, when different types of data structures are mapped to the local ontology. Mapping process starts from detecting standard cases in the database; every database component is transformed into adequate ontology component (e.g. a table to a class, a column to a property, a constraint to a relation). Some authors analyse methods for ontology transformation into conceptual data model, for example, in [20]. The ideas of various RDB2OWL transformations can be used for OWL2RDB transformation, but in many aspects they are different, because most of relational concepts may be mapped to ontology structures, but not every ontology concept may be directly mapped to a relational database.

We will try to describe and compare some of direct transformations from ontology to relational databases. We will use problems, formulated in [1], as criteria for comparing transformation approaches. So, the method of transformation is satisfactory, if it fills these conditions:

- The transformation should not lose data, data types and structure of ontology;
- The obtained relational database should be applicable, not only for saving all ontology information, but also practically and flexibly usable in information systems;
- The transformation should be fully automatic, realizable and have provable correctness.

We summarised some of transformation proposals in Table 1 trying to reveal advantages and shortcomings of them.

### Table 1. Methods of ontology to database transformation

| Method                            | Idea                                                                 | Advantages                                                                 | Shortcomings                                                                 |
|-----------------------------------|                                                                     |                                                                            |                                                                            |
| OWL2ER, ER2RDB [20]              | For transformation, conceptual graphs are used.                     | Formally defined transformation from ontology to conceptual model.          | Instead of direct transformation from OWL to RDB, two transformations are required: OWL to ER and ER to RDB; only main concepts are transformed. |
| Oracle Semantic Data Storage with inference engine [21] | Predefined and asserted facts are stored in database tables and can be directly accessed using SQL queries. For inference PL/SQL is used. | An industry-strength implementation. Unlike main-memory solutions there is no restriction on the amount of data it can handle. | Only OWL main constructs are handled. Missing constructs such as min/maxCardinality, oneOf, unionOf, and intersectionOf, limit its applicability for all applications. |
| Rule based transformation [1]     | Transformation is based on the set of rules that are applied to ontology to produce a relational database. | Many of OWL constructs are transformed to relational database concepts; check constraints are used for data type property restrictions. | Part of constructs (e.g. some property restrictions) will be lost during transformation. |
| OWL2DB algorithm [8]              | OWL data is mapped into tables and queries posed in the ontology language are translated to SQL queries. The transformation algorithm interprets OWL ontology like a graph. | The process autonomous, there is no human intervention once the input document is given. | The transformation is incomplete, just for saving class instances in relational database with all other semantics represented in OWL. |
| Storing Ontologies including Fuzzy Datatypes [2] | A schema to store ontologies with fuzzy datatypes into a database. Ontology and instances are stored in different schemas. | A schema structure can store ontologies and their instances in a FORDBMS capable for handling fuzzy data types. | The proposed schema covers only main constructs of OWL. |
| Large-Scale Ontology Management [15] | Existing data are not transformed but virtualized for the access from the ontology application. | Improves reasoning for ontologies by directly representing facts in RDB tables and representing action rules in SQL triggers. | Relational structure is not preserved, transformation is not fully automatic. |

The most similar to our approach is [1], where transformation of ontology to relational database is based on a set of rules called mapping rules that specify how to map constructs of the ontological model to the relational model. Mapping rules are specified on the model level, so they are applicable to any ontology that conforms to the ontological model. The disadvantage of this proposal is that some of the constructs in ontology will be lost when transforming the ontology to a relational database. Also, there are many other constructs in OWL ontology which are not considered in this approach, e.g. some types of property restrictions.
In [8], three possible approaches to store, manage and use the ontology data are presented. One is to build a special purpose database system which will be applicable to store and retrieve ontology specific data. The second is to exploit the rich data modelling capabilities of an object oriented database system. But the popularity of SQL and the efficiency of executing these queries make better using relational database systems. In this approach, OWL data are mapped into tables of a relational schema and queries represented in an ontology language are translated into SQL queries.

Proposed OWL to relational schema mapping system contains three parts – Ontology Modeller, Document Manager and Ontology Resource Manager. The transformation algorithm consists of 8 steps and interprets OWL ontology like a graph. The process is automatic; there is no human intervention once the input document is given. However, the transformation is very incomplete, just for saving class instances in relational database with all other semantics represented in OWL.

In [15], database-centric architectures are explored for storing and manipulating ontological data. Such solutions will take the advantage of existing standards for data management and the DBMS features like robustness, concurrency control, recovery, scalability. However, authors state that due to the inherent complexity of ontological queries, a straightforward database implementation of an ontology system may not perform efficiently, because database systems are not optimized for this type of applications. Authors introduce the solution where ontological information is directly represented in relational database tables and action rules are represented in SQL triggers. A single table called Fact Table is used for storing facts of ontology and a set of triggers which are fired when a fact is inserted, updated, or deleted from this table.

This solution is very straightforward and easy to use for adding or updating classes and querying. However, when using a single fact table, it becomes very large with many instances, and this affects performance. Another problem is using of existing data, which have to migrate to the single fact table. Authors propose to leaving the existing data in place and using the virtualised records for accessing them from the ontology application. This architecture consists of three layers: the relational database, the metadata necessary for the virtualization, and the interface for transparently accessing classes and instances.

This transformation method does not lose information, but it uses advantages of relational databases just for saving many records and does not preserve the real relational structure. The schema is in a low normal form and the performance of using transformed information is probably slow. Also, metadata for virtualization should be created manually, so this transformation process is not fully automatic.

In [2] authors deal with the need for managing large amounts of fuzzy data in the context of the Semantic Web. A schema to store ontology with fuzzy data types in a database is presented as the part of a framework designed to perform tasks of fuzzy information extraction and publishing. The database schema allows the storage of ontology along with its instances preserving all information. Ontology and instances are stored in different schemas in order to improve access to instances while retaining the capacity of reasoning over the ontology. Authors also present a brief description of the framework on which the database is included, and structures conforming the storage schema are proposed. The advantage of this proposal is that the obtained schema structure can store ontology and its instances in a FORDBMS capable for handling fuzzy data types, however, the method covers only main constructs of OWL.

6 Conclusions and Future Work

In this paper we presented mapping proposals for the transformation of ontology described in OWL to relational database schemas. According to these mapping, we created algorithms, transforming OWL concepts to RDB structures. We suggest that ontology classes should be mapped to relational tables, properties to relations and attributes, and constraints – to metadata. Using both direct mapping and metadata, we achieve applicable relational structure and avoid of losing the ontological data. A prototype tool, performing transformations, was implemented as add-in for ontology development tool Protégé, capable to importing OWL documents, and to exporting generated DDL scripts.

Currently, our algorithm is capable to transform most of OWL DL concepts, but our vision is to improve it and to propose mappings for most of OWL Full concepts. For example, currently our method does not save such ontological information as class complement, enumerated, intersection or union class. Also, we do not store information about some relations between properties, e.g. inverse functional, symmetric or transitive indication. We assume that all this information can be transformed into some structures of RDB or stored like metadata, but this needs further research. Also, we are preparing to extend our algorithms and tool according to forthcoming OWL 2 specification that strongly corresponds with current OWL but extends it by viewing the overall ontology as a set of axioms, allowing user derived data types, and other enhancements.

References


KNOWLEDGE MODELS REUSE FOR DATA ENGINEERING*

Justas Trinkunas, Olegas Vasilecas

Vilnius Gediminas Technical University, Information System Department, Sauletekio al. 11, LT-10223 Vilnius, Lithuania, justas@isl.vgtu.lt, olegas.vasilecas@fm.vgtu.lt

Abstract. Information systems are increasingly complex in particular with the enormous growth of the volume of data, different structures, different technologies and the evolutionary requirements of the users. Consequently, current applications require an enormous effort of design and development. The fast changing requirements are the main problem of creating and/or modifying conceptual data models. To improve this process we proposed to reuse already existing knowledge for conceptual modelling. In the paper we analysed reusable knowledge models. We presented our method for creating conceptual models from various knowledge models.

Keywords: knowledge models reuse, ontology, conceptual models

1 Introduction

We want to build information systems (IS) faster, cheaper and with fewer mistakes. How can we do it? One of the answers is – knowledge reuse. Most of information systems analyst at least one time considered how many times they have to analyse the same problems, create and design the same things. Most of them asked themselves - is it possible to reuse already existing knowledge? And we give the answer – yes. We believe that knowledge can be reused and even should be reused for IS development.

However even in these days most domain knowledge is elicited from documents, experts anew wasting previous efforts, time and resources. In this paper we present available knowledge source which could be reused in software engineering process and we present the new approach of building conceptual models from these knowledge sources. Real world domain knowledge or as we call domain ontology can bring outstanding benefits in software engineering.

We are proposing a method of knowledge reuse for those who are seeking an efficient and quality driven approach for data structures development, data integration strategies, enterprise data models, logical data models, database designs, data warehouse designs, or data mart designs.

The ontology (knowledge model) can be used for:
1. Data modelling [24],
2. Data integration [25],
3. Reverse engineering [25, 25],
4. Existing data models and DB validation [25],
5. Domain learning and teaching purposes [14].

The paper is organised as follows. The next chapter describes the theoretical background. Chapter 3 presents and analyse available knowledge sources which could be reused for conceptual data modelling. In chapter 4 we present the method for knowledge reuse for data modelling.

2 Theoretical background

In this chapter we present ontology and metamodel based transformations which are used in our proposed method. Also we present EER and OWL languages.

2.1 Ontologies and conceptual models

Many authors in their works propose different ontology definitions. We accept in [18] proposed ontology definition. Ontology defines the common terms and concepts (meaning) used to describe and represent an area of knowledge. An ontology can range in expressivity from a taxonomy (knowledge with minimal hierarchy or a parent/child structure), to a thesaurus (words and synonyms), to a conceptual model (with more complex knowledge), to a logical theory (with very rich, complex, consistent and meaningful knowledge).

Conceptual data models, also called semantic data models, were developed to capture the meaning of an application domain as perceived by its developers [27]. But there are the following main problems concerning conceptual modelling. Firstly, as discussed in [28], meanings of conceptual modelling constructs have to be

* The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project "Business Rules Solutions for Information Systems Development (VeTIS)" Reg. No. B-07042.
defined rigorously to employ them effectively. Often, however, rigorous definitions of these constructs are missing. Secondary, in general, most conceptual schemas are developed from scratch, which means wasting previous efforts and time [29]. Thirdly, domain knowledge acquired in the analysis of some particular domain is not used for conceptual modelling.

Because conceptual models are intended to capture knowledge about a real-world domain, the meaning of modelling constructs should be sought in models of reality. Accordingly, ontology, which is the branch of philosophy dealing with models of reality, can be used to analyse the meaning of common conceptual modelling constructs.

Companies install information systems to increase the effectiveness of activities, to get more profit and increase the value of the company. However to develop database today is hard and much time required work. When database designer creates new database he has to solve the same analytical problems every time:

- Every Application domain has a lot of terms and business rules which database designer has to understand.
- A database designer may not have knowledge of the domain in that is being modelled then thus must rely on the user to articulate the design requirements.
- One term can have many titles and meanings. This could be very aggravating circumstances for getting requirements for database.
- Database designer has to anticipate what will be the cycle of existence of the system, how it can change in the future, what are the threats and weaknesses of the system and how to avoid them.
- The fast changing requirements are the main problem of creating and/or modifying applications. Most of these requirements are related to business rules.
- In a case of change of application domain, database designer has to adopt the database quickly and effectively.

The advantage of using ontology for conceptual data modelling is the reusability of domain knowledge. As a result of it the conceptual data model will be made faster, easier and with fewer errors than creating conceptual data model in usual way.

2.2 Metamodel based transformations

The notion of model transformation is central to Model Driven Engineering. A model transformation takes as input a model conforming to a given metamodel and produces as output another model conforming to a given metamodel. A model transformation may also have several source models and several target models. One of the characteristic of a model transformation is that a transformation is also a model, i.e. it conforms to a given metamodel. The more information about metamodels can be found in [16].

Model Driven Architecture (MDA) [17] defines three viewpoints (levels of abstraction) from which some system can be seen. From a chosen viewpoint, a representation of a given system (viewpoint model) can be defined. These models are (each corresponding to the viewpoint with the same name): Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). MDA is based on the four-layer metamodeling architecture, and several OMG’s complementary standards. Layers are: metamodel (M3) layer, metamodel (M2) layer, model (M1) layer and instance (M0) layer.

In the paper we analyse ontology transformation to data model. The mapping from OWL (ontology web language) to EER was described in [18] document. However, this mapping is incomplete and it is not clear which elements from the OWL ontology are not transformed into data model. As a result of it some information from OWL ontology can not be used in data model. Metamodel based transformations are shown in figure 1.

![Figure 1. Metamodel based transformation](image-url)
2.3 **OWL and EER languages**

The OWL language provides mechanisms for creating all the components of ontology: concepts, instances, properties (or relations) and axioms. An EER represents the overall structure of an information system. It describes the conceptual relationships of different types of information rather than their physical structures. Ontology is much richer than EER (compare Table 1 and Table 2). However not everything from the ontology can be transformed into EER. So why we need to build knowledge models using OWL? The answer is simple – using richer OWL semantic we can build more precise domain models. Consequently transformed model is more precise too and of course the quality of the model is higher. The quality of the model is defined by these properties [19, 20]:

- **Legibility.** To measure legibility which expresses the ease with which a conceptual schema can be read.
- **Expressiveness.** A schema is said to be expressive when it represents user’s requirements in a natural way.
- **Simplicity.** A schema is said to be simple if it contains the minimum possible constructs.
- **Correctness.** Is used in a wide range of contexts leading to very different interpretations. A schema is syntactically correct when concepts are properly defined in the schema.
- **Completeness.** A schema is complete when it represents all relevant features of the application domain.
- **Understandability.** Understandability is defined as the ease with which the user can interpret the schema.

<table>
<thead>
<tr>
<th>Element name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWLOntology</td>
<td>Class</td>
</tr>
<tr>
<td>OWLClass</td>
<td>Class</td>
</tr>
<tr>
<td>ComplementClass</td>
<td>Class</td>
</tr>
<tr>
<td>EnumeratedClass</td>
<td>Class</td>
</tr>
<tr>
<td>DisjointClass</td>
<td>Class</td>
</tr>
<tr>
<td>IntersectionClass</td>
<td>Class</td>
</tr>
<tr>
<td>EquivalentClass</td>
<td>Class</td>
</tr>
<tr>
<td>RestrictionClass</td>
<td>Class</td>
</tr>
<tr>
<td>UnionClass</td>
<td>Class</td>
</tr>
<tr>
<td>Property</td>
<td>Property</td>
</tr>
<tr>
<td>OWLAnnotationProperty</td>
<td>Property</td>
</tr>
<tr>
<td>OWLOntologyProperty</td>
<td>Property</td>
</tr>
<tr>
<td>FunctionalProperty</td>
<td>Property</td>
</tr>
<tr>
<td>OWLDatatypeProperty</td>
<td>Property</td>
</tr>
<tr>
<td>OWLObjectProperty</td>
<td>Property</td>
</tr>
<tr>
<td>InverseFunctionalProperty</td>
<td>Property</td>
</tr>
<tr>
<td>SymmetricProperty</td>
<td>Property</td>
</tr>
<tr>
<td>TransitiveProperty</td>
<td>Property</td>
</tr>
<tr>
<td>OWLRestriction</td>
<td>Restriction</td>
</tr>
<tr>
<td>HasValueRestriction</td>
<td>Restriction</td>
</tr>
<tr>
<td>AllValuesFromRestriction</td>
<td>Restriction</td>
</tr>
<tr>
<td>SomeValuesFromRestriction</td>
<td>Restriction</td>
</tr>
<tr>
<td>CardinalityRestriction</td>
<td>Restriction</td>
</tr>
<tr>
<td>MaxCardinalityRestriction</td>
<td>Restriction</td>
</tr>
<tr>
<td>MinCardinalityRestriction</td>
<td>Restriction</td>
</tr>
<tr>
<td>OWLDataRange</td>
<td>DataType</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Element name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Model</td>
</tr>
<tr>
<td>Entity</td>
<td>Entities</td>
</tr>
<tr>
<td>Attribute</td>
<td>Attributes</td>
</tr>
<tr>
<td>AlternateKey</td>
<td>Attributes</td>
</tr>
<tr>
<td>ForeignKey</td>
<td>Attributes</td>
</tr>
</tbody>
</table>

Table 1. Main OWL elements

Table 2. Main EER elements
Available domain knowledge sources

In this section we review and analyse available knowledge sources which could be reused for conceptual data modelling. It is very important to have good quality domain knowledge source because transformation quality straightforward depends on knowledge quality. Of course transformed model can be and even should be improved by hands.

IS are increasingly complex in particular with the enormous growth of the volume of data, different structures, different technologies and the evolutionary requirements of the users. Consequently, current applications require an enormous effort of design and development. In fact, such applications require a detailed study of their fields in order to define their concepts and to determine the concepts’ relationships [10].

Knowledge models are reusable models, or in other words ‘templates’ to help jump start and/or quality assure data modelling efforts. They can be used to save time and costs on efforts to develop enterprise data models, logical data models, database designs and data warehouse efforts.

There are a lot of books and articles written about data modelling. However, most system professionals know how to model data. Actually that they need are reusable knowledge models which could be reused for the real projects and could save many hours of work [2].

Of course if we want to get high quality data model after transformation, knowledge model has to be simple, correct and complete. If knowledge model is not simple after the transformation we will get complex conceptual schema which has to be improved manually. If knowledge model is not correct after the transformation we will get the same mistakes in the conceptual model. If knowledge model is not complete (at least in our domain) after transformation we will get incomplete conceptual model.

We propose to improve knowledge model iteratively. All mistakes noticed in the conceptual model should be also rechecked in the knowledge model. Also if conceptual model is incomplete we have to add needed information into knowledge model. Step by step we can create sophisticated source of knowledge. We will not discuss the creation of ontologies in this paper because this topic needs bigger attention and is out of scope of this paper.

In the next chapters we analyse different knowledge sources which could be reused for conceptual data model building and we will try to evaluate which of them is the most suitable.

The knowledge sources can be classified into three main categories – commercial (for example IBM data model, industry data models [2]), freely available (for example SUMO and OpenCyc), manually created for specific purpose.

3.1 SUMO

In this section we present SUMO domain ontology which could be used as a knowledge model. The main difference between universal models and domain ontologies is that usually ontologies are tend to be abstract and to reuse them is more complicate than universal models. However, domain ontologies are really good knowledge source which could be reused.

Sumo is the largest free, formal ontology [1]. It contains more than 20000 terms and more than 70000 axioms when all domain ontologies are combined. Sumo consist of SUMO itself, the MId-Level Ontology (MILO), and domain ontologies (communications, countries and regions, distributed computing, economy, finance, engineering components, geography, government, military, North American industrial classification system, people, physical elements, transportation etc). SUMO is written in the SUO-KIF language.
SUMO ontology also provides check rules. It is really important that SUMO ontology is freely available for everyone. Everyone can participate in the ontology development and improvement process. The other advantage, ontologies provide a set of rules, i.e. we can restrict model. However most of these ontologies do not cover all domain area.

3.2 Cyc and OpenCyc

OpenCyc [5] is the open source version of the Cyc technology, the world's largest and most complete general knowledge base and commonsense reasoning engine. The entire Cyc ontology containing hundreds of thousands of terms, along with millions of assertions relating the terms to each other, forming an upper ontology whose domain is all of human consensus reality.

The Cyc project has been described as "one of the most controversial endeavours of the artificial intelligence history" [13], so it has inevitably garnered its share of criticism.

OpenCyc is similar to full Cyc, but its knowledge base is just a few percent of the full knowledge base and its functionality is greatly reduced. Since Cyc's success lies in the completeness of its knowledge base, the only people who really know the extent of Cyc's progress are Cycorp employees [4].

To add more, with trying to cover all the word, Cyc becomes so enormous large and abstract. The reuse of this ontology is very complicated. Some issues about Cyc reuse are discussed in [14].

3.3 Wikipedia based ontologies

In recent years few communities tried to extract structured information from Wikipedia. As a result of it Yago [7], DBPedia [8] and FreeBase [9] ontologies were created. In this section we shortly introduce these ontologies.

YAGO is a huge semantic knowledge base. According to [7] YAGO contains more than 2 million entities and 20 million facts about these entities. The authors of YAGO stated that YAGO has a manually confirmed accuracy of 95%.

DBPedia [8] is a knowledge base which allows you to make sophisticated queries against Wikipedia, and to link other data sets on the Web to Wikipedia data. The DBpedia data set currently provides information about more than 2 million entities. Altogether, the DBpedia data set consists of 218 million pieces of information (RDF triples). The accuracy of DBpedia is not confirmed at the moment.

Freebase [9] is an open, shared database that contains structured information on millions of topics in hundreds of categories. This information is compiled from open datasets like Wikipedia, MusicBrainz, the Securities and Exchange Commission, and the CIA World Fact Book, as well as contributions from our user community. The accuracy of Freebase is not confirmed at the moment.

YAGO, DBpedia, Freebase knowledge sources are be really valuable. At the moment DBpedia is the biggest Wikipedia based ontology. YAGO has confirmed accuracy. All of them are developed very quickly by means of amount, functionality, accuracy.

3.4 Industry data models

The book [2] provides a series of industry universal data models for each phase of an enterprise’s business cycle: people and organizations, products (services or physical goods), commitments which are established between people and/or organizations, transport shipment, work efforts, invoices, budgeting and accounting, human resources management and tracking.

An industry data model or universal data model is a model that is widely applied in some industry. The sufficiently effective industry data models were developed in the banking, insurance, pharmaceutical and automotive industries, to reflect the stringent standards applied to customer information gathering, customer privacy, consumer safety, or just in time manufacturing.

The authors of the book [2] claims that 60% of a data model (corporate or logical) or data warehouse design consists of common constructs that are applicable to most enterprises. This means that most data modelling or data warehouse design efforts are at some point recreating constructs that have already been built many times before in other organizations.

The authors provide nine subject areas: accounting and budgeting, human resources, invoicing and time billing, orders and agreements, people and organizations, product, shipments and deliveries, web and e-commerce, work effort and project management.

In addition, seven industry-specific Universal Data Models are available, including: banking, investments and financial services, healthcare, insurance, manufacturing, professional services, telecommunications, travel.
These universal data models are very useful for data modelling. However we can reuse only structure, these models do not contain any business rules which is also very important knowledge resource.

3.5 Commercial data models

There are many commercial data models which could be reused. We analysed well known IBM Data model M1. This M1 database contains the Banking Data Warehouse Model. The model is composed of entity relationship model for application development. It contains 910 entities and 5237 attributes.

Let’s examine the ‘product’ definition from IBM Data model.

Product identifies goods and services that can be offered, sold, or purchased by the financial institution, its competitors, and other involved parties, or in which the financial institution has an interest during the normal course of its business activity; for example, 'Product#220 (Xyz bank's private banking residential mortgage loan)', 'Product #988 (personal checking account)', 'Product #440 (securities trading service)'. 'Product' has 22 attributes and 28 relationships in the model. Small extraction from the model is provided in figure 2.

IBM data model has three layers – the abstract level, the middle level and model level. Such organisation of the model is very convenient for the reuse.

Commercial data models usually are very expensive and restricted by many means.

3.6 Other ontologies

Protégé [6] provides more than 50 domain ontologies however none of them can be used for conceptual data modelling because most of them contains only few dozens concepts and are totally immature.

WordNet is a large lexical database of English [15]. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. The resulting network of meaningfully related words and concepts can be navigated with the browser. WordNet is also freely and publicly available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing. WordNet can not be reused straightforward but it is very useful for finding synonyms and more abstract terms than constructing a model.

4 Proposed approach

In the previous section we analysed available sources which could be reused for conceptual modelling. In this section we describe our method for the knowledge reuse for conceptual modelling. We made few experiments, including the reuse of knowledge from already existing resources and from the recourses created manually.

Briefly we describe proposed method of building conceptual model from the reusable models. The method consists of these main steps:

1. Find or create appropriate knowledge source which can be used for transformation.
2. Build requirements using ontology.
3. The second step is the transformation of knowledge model into specific data model with our plug-in OntER. Created data model can be opened with Sybase Power Designer 12.0 tool and adapted for your needs.
4. The last step is the generation of physical data model with Power Designer 12.0 for a particular DBMS. This feature is already implemented in the original version of Power Designer 12.0.

Through a simple generation procedure, you can transfer the solid design framework of the conceptual data model to the physical data model. The physical data model adapts your design to the specifics of a DBMS and puts you well on the way to complete physical implementation.

Transformation process is shown in figure 2.

The first experiments were made from the domain ontology found in Protégé site [6]. The other experiments were made with ontology which was created by us. And finally we tried to experiment with other ontologies.
We created requirements metamodel with Eclipse tool (Figure 3). Also we used requirement model taken from [22]. This Requirements Model [21] is composed of a Functions Refinement Tree (Figure 4) to specify the hierarchical decomposition of the system, a Use Case Model to specify the system communication and functionality, and Sequence Diagrams to specify the required object-interactions that are necessary to realize each Use Case. The ontology is show in Figure 5.
After deep analysis of available knowledge sources we decided that the most completed ontology is CYC. The most expressive is SUMO. However the most suitable by all the quality properties listed in the previous chapter are universal models and commercial data models. The evaluation of the transformation showed that quality straightforward depends on the quality of knowledge model.

5 Conclusions and future works

The experiment results shows that proposed method is really effective. We analysed available knowledge sources which could be reused for conceptual data modelling. It is very important to have good quality domain knowledge source because transformation quality straightforward depends on knowledge quality. We believe that using our method models are created faster, cheaper and they are more reliable. However this method has few weaknesses: no tool support, small amount of reusable models, knowledge reuse methodology is not mature enough.

6 References


BUSINESS KNOWLEDGE EXTRACTION USING PROGRAM UNDERSTANDING AND DATA ANALYSIS TECHNIQUES

Bronius Paradauskas, Aurimas Laurikaitis
Kaunas University of Technology, Department of Information Systems,
Studentų St. 50, LT-51368 Kaunas, Lithuania,
parad@soften.ktu.lt, aurimaras@yahoo.com

Abstract. This article discusses the process of enterprise knowledge extraction from relational database data dictionary and data and source code of legacy information systems. Problems of legacy systems and main solutions for them are briefly described here. The use of database reverse engineering and program understanding techniques to automatically infer as much as possible the schema and semantics of a legacy information system is analyzed. Six step database reverse engineering algorithm for knowledge extraction from legacy systems is provided. Hypothetical example of knowledge extraction from legacy information system is presented.

Keywords: business knowledge extraction, relational database, database reverse engineering, legacy information systems.

1 The Problems of Legacy Information Systems

Legacy information system – any information system that significantly resists modification and evolution to meet new and constantly changing business requirements [2],[27]. Legacy systems typically contain incredible detailed business rules and form the backbone of the information flow of organization that consolidates information about its business [4]. A failure in one of these systems may have a serious business impact. Legacy information systems are currently posing numerous and important problems to their host organizations. The most serious of these problems are:

- systems usually run on obsolete hardware which is slow and expensive to maintain;
- maintenance of software is generally expensive; tracing faults is costly and time consuming due to the lack of documentation and a general lack of understanding of the internal workings of the system;
- integration efforts are greatly hampered by the absence of clean interfaces;
- legacy systems are very difficult, if not impossible, to expand.

In response to these problems, several approaches to change or replace legacy systems have been proposed. They are classified into the following three categories [25]: redevelopment, wrapping, and migration. Redevelopment involves process of developing system from scratch, using a new hardware platform, architecture, tools and databases. Wrapping involves developing a software component called wrapper that allows an existing software component to be accessed by other components. Migration allows legacy systems to be moved to new environments that allow information systems to be easily maintained and adapted to new business requirements, while retaining functionality and data of the original legacy systems without having to completely redevelop them.

Usually, in the process of replacing legacy systems the above three categories are combined in varying degrees. First thing needed when solving the problems of legacy systems is the exact picture of information. So reverse engineering must be accomplished in order to discover and extract as much as possible business knowledge from legacy sources.

2 Database Reverse Engineering

Database reverse engineering (DBRE) is defined as the application of analytical techniques to one or more legacy data sources to elicit structural information (e.g. term definitions, schema definitions) from the legacy information sources in order to improve the database design or produce missing schema documentation [15],[34].

Formally DBRE can be described as follows: Given a legacy database DBL defined as $(\{R_1,R_2,...,R_n\},D)$, where $R_i$ denotes the schema of the i-th relation with attributes $A_{i,1},A_{i,2},...,A_{i,m(i)}$, keys $K_{i,1},K_{i,2},...,K_{i,m(i)}$ and data $D_i=\{r_{i,1}(R_1),r_{i,2}(R_2),...,r_{i,n}(R_n)\}$, such that $r_i(R_i)$ denotes the data for schema $R_i$ at time $t$. Furthermore, $DB_L$ has functional dependencies $F=\{F_1,F_2,...,F_{k(i)}\}$ and reference and equality constraints $I=\{I_1,I_2,...,I_{l(i)}\}$ expressing relationships among the relations in $DB_L$. The goal of DBRE is to first extract $\{R_1,R_2,...,R_n\}, I, F$ and then use $I, F, D$ and $C_L$ (program code) to produce a semantically enhanced description of $\{R_1,R_2,...,R_n\}$ that includes, all relationships among the relations in $DB_L$ (incl. those that are implicit), semantic descriptions of the relations as well as the business knowledge that is encoded in $DB_L$ and $C_L$.

* The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project "VeTIS" (Reg.No. B-07042).
Since 1980, a wide range of DBRE methods have been published. All of them consist of extracting the conceptual schema from an operational legacy system. The conceptual schema can be expressed in some variant of the entity-relationship model or of the object-oriented model (ODMG, UML, OMT).

The approaches of Hainaut [13], Dumpala [10], Navathe [24], Casanova [6], Markowitz [23], Davis [9], Johannesson [22], Ramanathan [30] and Alhaij [1] focuses on transforming physical or logical schema to a conceptual schema. These methods require physical or logical schema to be supplied with all primary and foreign keys, some have heavy prerequisites on schema (3NF, meaningful names, etc.) so they can be applied only on ideally designed databases. These approaches use schema as the only input and extracted conceptual schema is not enriched with implicit constraints.

DB-MAIN [11], Premerlani [28], Signore [32], Petit [29], Chang [7], MeRCI [8], Valet [20], [21] methods uses data and program and DML code as input and extract explicit and implicit constructs. All of these approaches explore DDL code. The approaches of Premerlani and Chang examine data, Signore method analyze program code. DB-MAIN, Petit, MeRCI and Varlet methods analyze data and application code. Petit method is the only to extract functional dependencies from DDL code and program code, MeRCI method is the only to elicit redundancy from DDL code and program code. DB-MAIN method allows supplementing extracted specification with knowledge of analyst.

All DBRE approaches concentrate their research on some area: extracting some constraints and/or analyzing some information sources. No one from above mentioned methods proposes tools to automatically extract secondary identifiers, enumerated value domains, constraints on value domains, existence constraints and semantics of attributes.

This article presents VeŽI (Lithuanian: Veiklos žinių išgavimas – Business Knowledge Extraction) approach that semi-automatically discovers and extracts knowledge from legacy information systems [26], i.e. generates conceptual specification of legacy information system, including entities, attributes, relations, constraints, business rules, etc. VeŽI approach generates conceptual schema in Extended Entity-Relationship model. This model represents conceptual model constructs in graphical form; it is widely used in CAD tools. VeŽI approach is applied to relational databases only.

The basic schema information is extracted (Figure 1) from legacy information system database management system (DBMS). Then this schema information can be semantically enhanced using details from application code and data. The final conceptual schema is generated using the set of conceptual transformations.

![Figure 1. The Conceptual Diagram for Business Knowledge Extraction.](image)
3 The Approach for Extracting Business Knowledge

VeŽI approach includes two main database reverse engineering processes: data structure extraction and data structure conceptualization. An overview of VeŽI approach which is comprised of six main steps is shown in Figure 2.

The data structure extraction is the most crucial and difficult part of DBRE. Data structure extraction analyzes the existing legacy system to recover complete logical schema that includes implicit and explicit structures and constructs. Various program understanding and data analysis techniques are used to recover implicit structures and constraints. Data structure extraction consists of three main steps: data dictionary extraction, schema refinement and schema cleaning.

The data dictionary extraction is the simplest process of data structure extraction. It produces raw physical schema, which consists of declared data structures and constraints.

The schema refinement is the most important process of data structure extraction. The main problem of the data structure extraction is that most of structures and constraints were not explicitly defined, but instead implemented in procedural part of application. Many implicit constraints exist: primary and secondary identifier, reference and equality constraints, functional dependencies, meaningful names, etc. Schema refinement process enriches physical schema with implicit constructs and produces complete physical schema.

The schema cleaning removes or replaces physical constructs into logical ones and transforms complete physical schema into complete logical schema.

Figure 2. Conceptual View of the VeŽI Algorithm.

The data structure conceptualization interprets logical schema in a conceptual view and recovers conceptual schema. It detects and transforms or discards non-conceptual structures, redundancies, technical
optimizations and DMS-dependent constructs. Data structure conceptualization consists of three main steps: preparation, basic conceptualization and normalization.

The preparation discards dead data structures and technical data structures, i.e. prepares the schema, that it only contains structures and constraints that are necessary to understand the semantics of the schema.

The basic conceptualization extracts a basic conceptual schema without worrying about esthetical aspects of the result. Two kinds of transformation are used: untranslation and de-optimization. Though distinguishing between them may be arbitrary in some situations (some transformations pertain to both).

The conceptual normalization transforms various constructs and gives expressiveness, simplicity, minimality and extensibility of conceptual schema. It tries to make higher level semantic constructs explicit (e.g. is-a relation).

The Business Knowledge Encoding is a technical step that extracts knowledge in the form of an XML document.

Trivial example of legacy information system is provided in order to highlight each of the six steps and related activities outlined in Figure 2. Assume that the underlying legacy database \( DB_L \) is managed by a relational database management system. For simplicity, we assume without lack of generality or specificity that only the following relations exist in \( DB_L \), whose schema will be discovered using DBRE:

- **CLIENT** \([CID, \text{NAME}, \text{ADDR\_CITY, ADDR\_STREET, ADDR\_HOME\_NUM, ADDR\_ROOM\_NUM}]\)
- **PHONE** \([CID, \text{PHONE}]\)
- **CUSTOMER** \([CID, \text{CATEGORY}]\)
- **SUPPLIER** \([CID, \text{ACCOUNT}]\)
- **PRODUCT** \([PRODNUM, \text{NAME, PRICE, SUPPLIER}]\)
- **ORDER** \([CID, \text{ONUM, DOC\_DATE, OP\_DATE}]\)
- **DETAIL** \([CID, \text{ORDER, PRODUCT, PROD\_NAME, PROD\_PRICE, QUANTITY}]\)

In order to illustrate the code analysis and how it enhances the schema extraction, the following C code fragment is used representing a simple, hypothetical interaction with a legacy database:

```c
char *aValue, *cValue;
int bValue = 100;
........
/* more code */
........
EXEC SQL SELECT DOC\_DATE, OP\_DATE INTO :aValue,:cValue
FROM ORDER WHERE ONUM = :bValue;
........
/* more code */
........
if (*cValue < *aValue) 
{ cValue = aValue; }
........
/* more code */
........
printf("Document Date %s ", aValue);
printf("Operation Date %s ", cValue);
```

### 3.1 Data Dictionary Extraction

The goal of data dictionary extraction is to obtain the relational specifications from the legacy source. It is the simplest process of data structure extraction. Almost all CASE tools propose some kind of DDL code analysis for the most popular DMS. Some of them are able to extract relational specifications from the system data dictionary as well. VeZI approach uses Java Database Connectivity (JDBC) that is an API for the Java programming language that extracts database structure and properties: collections, entities, attributes, primary identifiers, foreign keys and indexes.

Result. Extracted raw physical schema is presented in Figure 3.
3.2 Schema Refinement

The goal of most important process of data structure extraction is to identify and extract structures and constraints that were implemented implicitly or were discarded during application development. Many implicit constraints exist: primary and secondary identifier, reference and equality constraints, functional dependencies, meaningful names, etc.

3.2.1 Data Analysis

The goal of data analysis is to augment raw physical schema with constraints that were not discovered during data dictionary extraction:

- Finding optional attributes. Optional constraints of attributes were obtained during data dictionary extraction process, but there still exist optional attributes that were declared as mandatory attributes.
  - There are some values of attribute „CLIENT.ADDR_ROOM_NUM“ with space symbol, so this attribute is appended to the list of possible optional attributes.
- Finding enumerated value domains. Many attributes must draw their values from a limited set of predefined values.
  - There are very few distinct values of attribute „CUSTOMER.CATEGORY“, so possible values of this attribute are noted.
- Finding constraints on value domains. In most DMS, declared data structures are very poor as far as their value domain is concerned. Quite often, though, strong restriction is enforced on the allowed values.
  - The minimal value of attribute „PRODUCT.PRICE“ is 0 and the maximum value is more than 0, so valid value domain is positive numbers and zero (0).

Following data analysis, if possible optional attributes were identified; list of these attributes with possible nullable values must be presented to the user for a final decision.

Result. Physical schema with additional attribute constraints is presented in Figure 4.
3.2.2 Abstract Syntax Tree Generation

The goal of generation of abstract syntax tree (AST) for the legacy application code is to support various techniques of program understanding for discovery of implicit structures and constraints. Lexical analyzers and parsers are used for AST generation (AST is described in [5], [33]). The AST will be used by few steps of data structure extraction.

Result. Abstract syntax tree of trivial example is presented in Figure 5.

3.2.3 Identifier Extraction

The goal of identifier extraction is to obtain primary identifiers, if primary key information was not retrieved from data dictionary, and discover secondary identifiers.

The algorithm of primary identifier extraction proceeds as follows: for each entity, it first identifies the set of candidate attributes, which are mandatory and unique attributes. If there is only one candidate attribute per entity, then that attribute is the primary identifier. Otherwise, AST is analyzed for rule-out patterns and those attributes are eliminated from the candidate set, which occur in the rule-out pattern. By definition of primary key, this rules out the possibility that the attributes \( a_1, \ldots, a_n \) form a primary key. Three sample rule-out patterns are:

1. \text{SELECT DISTINCT <selection> FROM <table>}

Figure 4. Physical Schema with Attribute Constraints.

Figure 5. Abstract Syntax Tree for the Legacy Application Code.
WHERE a1=<expr1> AND a2=<expr2> AND ... AND an=<exprn>

2. SELECT <selection> FROM <table>
WHERE a1=<expr1> AND a2=<expr2> AND ... AND an=<exprn>
GROUP BY ...

3. SELECT <selection> FROM <table>
WHERE a1=<expr1> AND a2=<expr2> AND ... AND an=<exprn>
ORDER BY ...

Following AST, if a primary identifier cannot be identified, the reduced set of candidate attributes must be presented to the user for a final primary identifier selection.

If unique indexes for mandatory attributes exist which are not primary identifiers, then these attributes are marked as secondary identifiers.

Result. Primary and secondary identifiers are as follows (Figure 6):

- Entity „SUPPLIER” has only one candidate attribute (mandatory and unique), so „SUPPLIER.CID” is a primary identifier.
- Entity „PRODUCT” has two candidate attributes „PRODUCT.PRODNUM” and „PRODUCT.NAME”. No one attribute occur in rule-out pattern so the user must select primary identifier. The user chooses attribute „PRODUCT.PRODNUM” as a primary identifier.
- Attribute „PRODUCT.NAME” of entity „PRODUCT” is a secondary identifier, because it is unique and mandatory attribute which is not a primary identifier.

Figure 6. Physical Schema with Identifiers.

3.2.4 Reference and Equality Constraints Extraction

The goal of reference and equality constraints extraction is to identify constraints to help classify the extracted entities, which represent both the real-world entities and the relationships among them. This is done using reference and equality constraints, which indicate the existence of inter-relational constraints including class/subclass relationships.

Reference constraint is described as follows: Let R1 and R2 be 2 entities, and A and B be attributes or set of attributes of R1 and R2 respectively. Reference constraint R1.A<<R2.B denotes that a set of values appearing in R1.A is a subset of R2.B (R2.B must be primary or secondary identifier). Reference constraints are discovered by examining all possible subset relationships between any two entities R1 and R2. Reference constraints can be identified in an exhaustive manner as follows: for each pair of entities R1 and R2 in the legacy source schema, compare the values for each non-key attribute combination A in R1 with the values of each
primary (or secondary) identifier combination \( B \) in \( R2 \) (note that \( A \) and \( B \) may be single attributes). Reference constraint \( R1.A \ll R2.B \) may be present if:

1. \( A \) and \( B \) have same number of attributes.
2. \( A \) and \( B \) must have pair wise domain compatibility (matching data types and matching maximum length of attributes).
3. \( R1.A \subseteq R2.B \).

If there is additional subset dependency \( R2.B \subseteq R1.A \), then equality constraint may exist.

Information for reference and equality constraints extraction is obtained from three sources: physical schema with identifiers provides entities, attributes, primary and secondary identifiers and foreign keys (if available), equi-join query finder explores AST and provides pairs of entities and corresponding attributes, which occur together in equi-join queries in AST (the fact that two entities are used in a join operation is evidence for the existence of an reference or equality constraint between them), and subset dependency explorer run SQL queries against legacy database.

In order to check the subset criteria (3), the following generalized SQL query templates are provided, which are instantiated for each pair of primary (or secondary) identifier of one entity and attribute of the other entity combinations and run against the legacy source:

\[
C1 = \text{SELECT COUNT (\ast) FROM R1 WHERE A NOT IN (SELECT B FROM R2)}
\]

\[
C2 = \text{SELECT COUNT (\ast) FROM R2 WHERE B NOT IN (SELECT A FROM R1)}
\]

If \( C1 \) is zero, we can deduce that a reference constraint \( R1.A \ll R2.B \) may exist, likewise, if \( C2 \) is zero that a reference constraint \( R2.B \ll R1.A \) may exist. Note that it is possible for both \( C1 \) and \( C2 \) to be zero. In that case, we can conclude that the two sets of attributes \( A \) and \( B \) are equal, so equality constraint may exist.

Result. Reference and equality constraints are as follows (Figure 7):

\[\text{REF=} \{ \text{PHONE.CID<<CLIENT.CID, CUSTOMER.CID<<CLIENT.CID, ORDER.CID<<CUSTOMER.CID, SUPPLIER.CID<<CLIENT.CID, PRODUCT.SUPPLIER<<SUPPLIER.CID, DETAIL.PRODUCT<<PRODUCT.PRODNUM} \}, \text{EQU=} \{ \text{DETAIL.CID, DETAIL.ORDER<<ORDER.CID, ORDER.ONUM} \}.\]

![Figure 7. Physical Schema with Referential Constraints.](image-url)
3.2.5 Other Constraints Extraction

The goal of this step is to obtain implicit constructs, i.e. constraints and dependencies: coex, exact-1, disjoint(R1.A1,R2.A1), R1 \cup R2.A1 \rightarrow A2, R1.A1 \in (R2.A1 \cup R3.A1), R.A1 = \rightarrow (BA1 = A1) and redundancy constraint rd. In order to find constraints and dependencies, the generalized SQL query templates are used and run against the legacy source.

For example, in order to discover coex constraint, the following generalized SQL query template is used, which is instantiated for each pair of the entity optional attribute combinations and run against the legacy source:

\[ C = \text{SELECT COUNT (*) FROM R WHERE NOT((A IS NULL AND B IS NULL) OR (A IS NOT NULL AND B IS NOT NULL))} \]

If C1 is zero, we can deduce that coex constraint between A and B exist, likewise, if C is greater than zero then coex constraint between A and B doesn’t exist.

Result. Other constraints are as follows (Figure 8):
- disjoint(CUSTOMER.CID, SUPPLIER.CID);
- \( \text{CLIENT.CID in (CUSTOMER.CID} \cup \text{SUPPLIER.CID}) \);
- rd(DETAIL.PROD_NAME << PRODUCT.NAME).

3.2.6 Code Analysis

The objective of code analysis is twofold: (1) augment entities with domain semantics, and (2) identify business rules and constraints not explicitly stored in the database, but which may be important to the process of reverse engineering. This approach to code analysis is based on program understanding, which includes slicing [3], [18], [19] and pattern matching [31].

The mining of semantic information from source code assumes that in the application code there are output statements that support report generation or display of query results. From output message string that usually describes a displayed variable v, semantic information about v can be obtained. This implies location (tracing) of the statement s that assigns a value to v. Since s can be associated with the result set of a query q, we can associate v’s semantics with a particular attribute of entity.
The first step is the construction of system dependency graph (SDG) from abstract syntax tree. SDG is constructed in four stages [16]: 1) augmented control flow graph construction (ACFG) from the AST; 2) computation of the post dom graph from ACFG; 3) construction of the program dependency graph (PDG) using ACFG and the post dom graph; 4) construction of the SDG using the PDG. The PDG for the AST of Figure 5 is presented in Figure 9.

![Figure 9. System Dependency Graph for the Legacy Application Code.](image)

The next step is the pre-slicing. From the AST all the nodes are identified corresponding to input, output and embedded SQL statements. If an identifier node (which corresponds to the occurrence of a variable in that statement) exists in the subtree of that statement node, then the actual variable name is appended to the list of slicing variable. For example, for the AST in Figure 5, the array contains the following information depicted in Table 1.

<table>
<thead>
<tr>
<th>Slicing Variable</th>
<th>Type of Statement</th>
<th>Direction of Slicing</th>
<th>Text String (Only for Print Nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aValue</td>
<td>Output</td>
<td>Backwards</td>
<td>“Document Date”</td>
</tr>
<tr>
<td>cValue</td>
<td>Output</td>
<td>Backwards</td>
<td>“Operation Date”</td>
</tr>
</tbody>
</table>

For each slicing variable identified by the pre-slicing step, code slicing and analysis are performed on the AST. In the above example, the slicing variables that occur in SQL and output statements are aValue and cValue. The direction of slicing is fixed as backwards or forwards depending on whether the variable in question is part of an output (backwards) or input (forwards) statement. The slicing criterion is the exact statement (input or output) vertex that corresponds to the slicing variable.

During code slicing step the flow and control edges of the PDG for the source code are followed for each slicing variable and only those vertices are retained that were reached by traversal. The result of slice consists of the set of vertices and the set of edges induced by this vertex set that are relevant to the slice with respect to the slicing variable. Figure 10 shows backward slice for the SDG in Figure 9 with respect to printf(cValue) vertex. The reduced AST that correspond the SDG in Figure 10 is shown in Figure 11.
During the analysis, information shown in Table 2 is extracted, while traversing the reduced AST.

1. *dcln* node contain information about data type of the identifier.
2. *embSQL* node contain the mapping information of identifier name to corresponding attribute and entity.
3. *print/scanf* nodes contain the mapping information from the text string to the identifier. In other words we can extract the meaning of the identifier from the text string.

Table 2. Information Inferred During Analysis Step.

<table>
<thead>
<tr>
<th>Identifier Name</th>
<th>Meaning</th>
<th>Possible Business Rule</th>
<th>Data type</th>
<th>Column Name</th>
<th>Table Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>aValue</td>
<td>Document date</td>
<td>if (*cValue &lt; *aValue) { cValue = aValue; }</td>
<td>Char * =&gt; string</td>
<td>DOC_DATE</td>
<td>ORDER</td>
</tr>
<tr>
<td>cValue</td>
<td>Operation date</td>
<td></td>
<td>Char * =&gt; string</td>
<td>OP_DATE</td>
<td>ORDER</td>
</tr>
</tbody>
</table>

It is important to note, that business knowledge is identified by matching templates against code fragments in the AST. So, patterns for discovering business rules must be developed which are encoded in loop structures and/or conditional statements and mathematical formulae, which are encoded in loop structures and/or assignment statements.

### 3.3 Schema Cleaning

The goal of schema cleaning is to remove or replace physical constructs into logical ones and transform complete physical schema into complete logical schema. All the physical constructs can be discarded at this point because they do not provide any information about the database logical structure. The main transformations are:

- Removing indexes.
• Removing collections. If collection includes only one entity, then the name of collection is denoted as alternative name of entity.

Result. Schema after removal of indexes and collections is presented in Figure 12.

![Figure 12. Complete Logical Schema.](image)

### 3.4 Preparation

The goal of preparation is to prepare the schema that it only contains structures and constraints that are necessary to understand the semantics of the schema. The technical data structures are discarded. The names of the objects are given by the programmers, who have used some naming rules. Now the names can be changed to give more information on the named objects:

• Removing common prefixes. A common naming conversion consists in prefixing each attribute name by the name (or a short name) of the entity type. Those prefixes do not give any information, so they can be removed.

Result. Changes to attribute names are as follows (Figure 13): attribute „PRODUCT.PRODNUM“ name become „PRODUCT.NUM“, „ORDER.ONUM“ – „ORDER.NUM“, „CLIENT.CID“ – „CLIENT.ID“. The name of attribute „PHONE.PHONE“ is not modified, because it completely match the name of entity.
3.5 Basic Conceptualization

The goal of basic conceptualization is to extract a basic conceptual schema without worrying about esthetical aspects of the result. Two kinds of transformation are used: untranslation and de-optimization. Though distinguishing between them may be arbitrary in some situations (some transformations pertain to both). A complete description of transformations can be found in [12], [14], [17]. The main transformations are:

- Removing redundancy.
- Transforming foreign key into relationship type and internal constraint.
- Transforming foreign key into relationship type.
- Transforming list of attributes into a multivalued attribute.
- Aggregating attributes.

Result. Transformations are as follows:

- Redundancy $rd(DETAIL.PROD_NAME << PRODUCT.NAME)$ and its attribute "DETAIL.PROD_NAME" are removed.
- Foreign keys "CUSTOMER.CID", "SUPPLIER.CID" and constraints $disjoint(CUSTOMER.CID, SUPPLIER.CID)$, $CLIENT.IDin (CUSTOMER.CID \cup SUPPLIER.CID)$ are transformed into relationship types "customer" and "supplier" respectively. General entity "CLIENT" includes new constraint exact-1. Schema after this transformation is presented in Figure 14.
- Foreign keys "PHONE.CID", "ORDER.CID", "DETAIL.PRODUCT" , "PRODUCT.SUPPLIER" are transformed into relationship type "phone", "order", "detail" , "product" respectively.
- Equality constraint "DETAIL.CID, DETAIL.ORDER" is transformed into relationship type "detail1".
- Attributes "CLIENT.ADDR_CITY", "CLIENT.ADDR_STREET", "CLIENT.ADDR_HOME_NUM", "CLIENT.ADDR_ROOM_NUM" are aggregated to compound attribute "ADDR". Schema after transformations of basic conceptualization is presented in Figure 15.

![Figure 13. Prepared Logical Schema.](image-url)
3.6 Normalization

The goal of normalization is to improve the expressiveness, the simplicity, the readability and the extensibility of the conceptual schema. It tries to make higher level semantic constructs explicit (e.g. is-a relation). A complete description of transformations can be found in [12], [14], [17]. The main transformations are:

- Merging entity types.
• Transforming entity type into attribute.
• Transforming entity type into relationship type.
• Transforming relationship types into multi-domain role.
• Transforming relationship types into is-a relation.
• Connecting entity types with is-a relation. If is-a relation of type disjoint or partition is found, it must be presented to the user for a final validation.
• Name processing. Some names can be changed to be more meaningful. For example, the attribute “TOT” can be changed to “TOTAL”, “ADDR” – “ADDRESS”. A usual naming rule is to use lowercase for relationship type names, uppercase for the entity type names and capitalized for the attribute names.

Result. Transformations are as follows:
• Entity type „PHONE“ is transformed into attribute „CLIENTPHONE“.
• Entity type „DETAIL“ is transformed into relationship type „detail“. Schema after this transformation is presented in Figure 16.
• Relationship types „customer“, „supplier“ are transformed into is-a relation of type partition and constraint exact-1 of entity „CLIENT“ is removed. Schema after this transformation is presented in Figure 17.
• Attribute „CLIENT.ADDR“ is renamed into „CLIENT.ADDRESS“. Attribute names are capitalized. After this transformation final conceptual schema is composed (Figure 18).

![Diagram of conceptual schema after entity types transformation into attribute and relationship type.](image-url)
3.7 Business Knowledge Representation

Six steps of DBRE enable the extraction of the following schema information from the legacy database:

- Entities;
- Compound, multivalued and atomic attributes;
- Attribute constraints;
- Primary and secondary identifiers;
• Relationships;
• Is-a relations;
• Multi-domain roles;
• Business rules.

A conceptual overview of the extracted schema is represented by extended entity-relationship diagram shown in Figure 18.

4 Conclusion

Legacy information systems contain incredible detailed business rules and form the backbone of the information flow of organization, but their maintenance is very expensive and it is very difficult, if not impossible, to expand them. Reverse engineering is the essential part of process of changing and replacing legacy systems. Its main objective is to discover and extract business knowledge from legacy sources. Reverse engineering builds the powerful foundation for renovation of IT systems that enables the application of new technologies and programs.

Database reverse engineering algorithm provided in this article includes two main database reverse engineering processes: data structure extraction and data structure conceptualization. The data structure extraction is the most crucial and difficult part of DBRE. Data structure extraction analyzes the existing legacy system to recover complete logical schema that includes implicit and explicit structures and constructs. Various program understanding and data analysis techniques are used to recover implicit structures and constraints. The data structure conceptualization interprets logical schema in a conceptual view and recovers conceptual schema. It detects and transforms or discards non-conceptual structures, redundancies, technical optimizations and DMS-dependent constructs.

Six steps of database reverse engineering algorithm enable the extraction of entities, compound, multivalued and atomic attributes, attribute constraints, primary and secondary identifiers, relationships, as-a relations, multi-domain roles and business rules. This knowledge then could be used when changing or replacing legacy systems, i.e. when redeveloping, wrapping, or migrating legacy systems.

References


DEFINING WELL-FORMEDNESS CONSTRAINTS WITH OCL*

Kestutis Normantas, Olegas Vasilecas, Sergejus Sosunovas

Vilnius Gediminas Technical University, Information Systems Research Laboratory, Saulėtekio al. 11, LT20221, Vilnius, Lithuania, {kestutis.normantas,olegas,sergejus}@isl.vgtu.lt

Abstract. Metamodel well-formedness takes very important place in Model-Driven Architecture (MDA). Object-Role Modeling (ORM) as candidate to MDA computation independent level standard should be well-formed and compliant to MDA standards. In this paper we present formal specification of ORM well-formedness rules expressed in OCL as well as method for implementation it in MDA based modelling tool. Formal specification is implemented in prototype tool thus showing feasibility of proposed method.

Keywords: MDA, ORM, metamodel, well-formedness rules, OCL

1 Introduction

Recently, much attention is paid on Model Driven Architecture (MDA) software development approach. The primary goals of MDA are portability, interoperability, and reusability through an architectural separation of concerns between the specification and implementation of software [1]. MDA provides a systematic framework to understand, design, operate, and evolve all aspects of enterprise systems, using engineering methods and tools. The framework is based on modelling different aspects and levels of abstractions of such systems, and exploiting interrelationships between these models [2].

Object Role Modeling (ORM) [3] is a fact-oriented, attributes-free modelling approach divergent from tradition modelling approaches by its expressive power to capture all domain related details in the conceptual system model. It is widely adopted that rich ORM notation in combination with verbalization mechanism is suitable to express business rules in form understandable both to technicians and business experts. Recently, ORM is being considered as possible standard for business rules expression at the computation independent model (CIM) level within the Business Rules Special Interest Group formed within the Object Management Group (OMG). A standard metamodel of ORM is essential for this modelling approach to be a MDA CIM level modelling standard. Such metamodel would facilitate the interchange of ORM model data between different software tools [1], and enable transformations between ORM and other modelling languages within MDA.

Unfortunately, there are not made much research on this topic. MDA compliant ORM metamodel is discussed by Halpin in [1]. A Meta Object Facility (MOF) metamodel for ORM is suggested in his paper. However, discussed metamodel is not formal enough to be used within MDA. Provided metamodel does not contains specified well-formedness rules, which could be applied to check syntactically ORM model, though author has formalized some of them in the formalization of ORM presented in [3].

A MOF metamodel for ORM presented by Sosunovas and Vasilecas in [5],[6],[7],[8] slightly differs from the one mentioned above; however, all fundamental ORM notation elements are involved. Reluctantly, authors do not take attention to well-formedness of ORM, because the main aim is a conversion of ORM model to the Object Constraint Language (OCL) expressions. Much attention to well-formedness of ORM is given in Jarrar publication [9]. Author presents a number of ORM schemas satisfiability checking (well-formedness) patterns, and discusses them regarding those described in [3]. However, MDA compliant metamodel is not considering in his research.

In this paper we present an MDA based approach of specification and implementation of ORM well-formedness rules in MDA modelling tool. The paper is structured as follows: in Section 2, specification of ORM well-formedness rules with OCL is presented. In Section 3, a discussion on implementation of these rules by presenting architecture of prototype tool and describing implementation process is made. In Section 4, conclusions are made after discussion on related research.

2 Specification of ORM Well-Formedness Rules with OCL

The MDA is progressively becoming an important aspect of software development. MDA is a standard framework adopted by the Object Management Group (OMG). According to [17], it allows developers to link object models together to build complete systems and prevents design decisions from being intertwined with the application and keeps it independent of its implementation.

* The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project “Business Rules Solutions for Information Systems Development (VeTIS)” Reg. No. B-07042
Metamodels within MDA plays significant role [18]: they specify concepts of language used for specification of models. In order to use a modelling approach within MDA, it needs to be defined in appropriate level of formality. The MOF is adopted by OMG as a standard that defines the language used to define modelling languages [2].

The MOF metamodel for ORM used in this research is presented and described in details in [5]. A MOF metamodel is an information model that defines the abstract syntax of a particular modelling language. The purpose of an abstract syntax model is to describe the concepts in a language and the relationships that exist between those concepts [10]. An abstract syntax model should also describe the rules by which a model written in the language is deemed to be well-formed, i.e. is syntactically valid. Well-formedness rules are particularly useful when it comes to implementing a tool to support the language as they can be used to validate the correctness of models as they are created [10].

The OCL is a part of UML, another important software development standard relevant to MDA. OCL facilitates writing queries on a UML model providing possibility to specify UML models in unambiguous manner. Because MOF is based on UML infrastructure, OCL is suggested [16] to be used within MDA for precise specifications of metamodels as well.

We assume that the reader is familiar with ORM and OMG provided standards; therefore we will not make an introduction on ORM and OMG standards. Otherwise, more information about ORM could be found in [11][12][13][14], related OMG standards in [15][16][17][19].

According to [3], well-formedness rules could be abstracted into two categories:

• Inappropriate usage of constraints
• Contradiction between different types of constraints.

These categories will be explained in the following chapters. We will not provide formal foundation for ORM well-formedness rules since it had been already described by Halpin in [3]. Moreover, refinement, optimization, and re-representation of these rules had been proposed by Jarrar [9]. Jarrar has shown that some of them may be satisfied by extending unsatisfiable schemas to satisfiable. However, they are fundamental aspect for refinement of ORM well-formedness. Therefore, we concentrate only on applying ORM well-formedness rules to MOF metamodel for ORM and implementation of this metamodel in MDA based modelling tool. A subset of ORM well-formedness rules specified with the OCL will presented in the next subsections.

2.1 Inappropriate usage of constraints

Inappropriate usage of constraints involves cases when application of constraints is controversial. For example, adding frequency constraint to every predicate’s role leads to redundancy in population of these roles instances [9]. Respective rule is formed as follows [3]:

Rule 1: A frequency constraint cannot span a whole predicate.

In Figure 1 related metamodel fragment is presented. ORM predicate may consist of one or more roles, which may be restricted by the role constraint through the role reference. The frequency constraint with correspondent lower and upper bounds is a subtype of the role constraint. Thus, a predicate instance accesses relative instance of frequency constraint over associated role reference instance.

Figure 1. Metamodel fragment presenting reference between role and frequency constraint

Corresponding to the first rule OCL expression is formed accordingly:

```oclasses
context Predicate
inv spanOfAllPredicateByFrequencyConstraint:
  self.role.roleReference.theRoleReference->
  selectoclIsTypeOf(FrequencyConstraint))->notEmpty() implies
  self.role.roleReference.theRoleReference->
  selectoclIsTypeOf(FrequencyConstraint))->size()<self.role->size()
```
The invariant above states that for every predicate instance, if the set of predicate's roles restricted by the frequency constraint is not empty, then the number of related role instances must be greater than number of role instances restricted by frequency constraint.

Another related to this category rule reflects syntactically correct formation of ORM schema. The rule is formed as follows [3]:

Rule 2: A frequency constraint of 1 is never used (the uniqueness constraint must be used instead).

A metamodel fragment in Figure 1 describes this rule. Rule states that instead of using frequency constraint with lower and upper bound of 1, uniqueness constraint must be used. Conformable to the second rule OCL expression is formed in this manner:

context FrequencyConstraint
inv frequencyConstraintOfOneOrZero:
  self.theRoleReference->notEmpty() implies
  not (self.lower = 0 and self.upper = 0)
  and
  not (self.lower = 1 and self.upper = 1)

The invariant above denotes that existence of frequency constraint instance in the model (it is expressed as test for the collection of related role reference objects to be not empty) implies upper and lower bound not to be of value 1. Additionally, check for value of 0 is supplemented to this invariant.

The third rule from this category indicates overlapping of uniqueness constraints to the same predicate's roles. From modeling perspective overlapping constraint contradicts to the one which is overlapped. Considering example presented in Figure 2, object type A must play role r1 at most one time, as it is restricted by uniqueness constraint (added above the role r1), but uniqueness constraint spanned roles r1 and r2 indicates that roles r1 and r2 can be played many times by object types A and B respectively (cardinality many to many).

Figure 2. Overlapping of uniqueness constraints

The rule is formed as [3]:

Rule 3: No uniqueness constraint can be spanned by a longer uniqueness constraint.

In the metamodel fragment (Figure 3), the role constraint of uniqueness constraint type is emphasized. Uniqueness constraint may be type of external or internal uniqueness constraint. Internal constraint applies to roles involved in one predicate as it was shown in the example above; external uniqueness constraint applies to roles involved in different predicates denoting that combination of roles instances must be unique in population of constrained roles instances.

Figure 3. Metamodel fragment presenting possible uniqueness constraint types

Corresponding to the third rule OCL expression is formed as follows:
context Role
inv spanOfUCbyLongerUC:
  self.roleReferences->notEmpty() implies
  self.roleReferences->
  forall (r1,r2| r1<>r2 implies 
    r1.theRoleConstraint.oclIsTypeOf(UniqueConstraint) 
    and 
    r2.theRoleConstraint.oclIsTypeOf(UniqueConstraint))
The invariant above indicates that for every different pair of role instances restricted by unique constraint the number of restricted instances of role must be greater than the number of another one. Hence, the number of restricted roles by uniqueness constraint is prevented to be greater than a number of restricted roles by another uniqueness constraint within predicate.

2.2 Contradiction between different types of constraint

ORM supports many constraint types [11]; hence, contradiction in application of different constraints to roles may arise. For example, incompatible application of frequency constraint to predicate’s roles spanned by the uniqueness constraint. In Figure 4, frequency constraint denoting that role r2 cannot be played more than 10 times contradicts with uniqueness constraint indicating that roles r1 and r2 can be played many times by object types A and B respectively (cardinality many to many).

Figure 4. Contradiction between frequency constraint and uniqueness constraint

According to [3], correspondent rule is formed as:

Rule 4: No role sequence exactly spanned by a uniqueness constraint can have a frequency constraint.

Respective to the rule 4 OCL expression is formed as follows:

context Predicate
inv overlapOfUniquenessConstraintByFrequencyConstraint:
  self.role.roleReferences.theRoleConstraint->select(oclIsTypeOf(UniqueConstraint))->notEmpty() and
  self.role.roleReferences.theRoleConstraint->select(oclIsTypeOf(FrequencyConstraint))->notEmpty() implies
  self.role.roleReferences.theRoleConstraint->select(oclIsTypeOf(UniqueConstraint))->size() >
  self.role.roleReferences.theRoleConstraint->select(oclIsTypeOf(FrequencyConstraint))->size()

The rule above states that if all roles are restricted by uniqueness constraint within one predicate then the frequency constraint cannot be applied to them. Additionally, test for existence of uniqueness constraint as well as frequency constraint references to predicate’s roles are supplemented.

Another case of constraints contradiction is contradiction between exclusion and mandatory constrains. This type of rule applies only to those constraints which restricts roles played by the same object type. Figure 6 presents schema where object type A must play role r1 (denoted with mandatory constraint), while exclusion constraint between roles r1 and r3 indicates, that either role r1 or role r3 must be played by object type, but not both. Thus, population of role’s r3 instances will never exist.
Related well-formedness rule is formed as follows [3]:

Rule 5: An exclusion constraint cannot be specified between roles if at least one of these roles is marked as mandatory.

Figure 6. Contradiction between exclusion and mandatory constraints

ORM metamodel fragment regarding Rule 5 is presented in Figure 7. The role constraint as well as mandatory constraints is subtype of constraint. The set constraint is a subtype of role constraint and may be one of set types enumerated in SetConstraintKind. The instance of role accesses relative mandatory constraint over associated constraint instance and accesses set constraint over associated role reference and role constraint instances.

Figure 7. Metamodel fragment presenting reference between set constraint and mandatory constraint

Corresponding to the Rule 5 OCL expression is formed as follows:

```ocl
context Role
inv contradictionOfMandatoryAndExclusionConstraints:
    self.constraint->select(oclIsTypeOf(Mandatory))->notEmpty() implies
    self.roleReferences.theRoleConstraint->forAll(c:RoleConstraint|
        c.oclIsTypeOf(SetConstraint) implies
        c.theRoleReference.theRoleConstraint.oclAsType(SetConstraint) ->forAll(sc:sc.setTpype<>SetConstraintKind::exclusion)
    )
```

The invariant above denotes that if role instance is restricted by mandatory role constraint, then for every role instance restricted by mandatory constraint, restriction by set constraint kind of exclusion is not possible. It should be noticed, that compatibility of set comparison constraint kind of inclusiveOR or exclusiveOR in some cases is negotiable as well.

Another rule of this category should be invoked when exclusion constraint is applied to roles of object types where one is subtype of another. Figure 8 presents schema where object type C is a subtype of object type B. Depending on exclusion constraint between roles r3 and r4, the population of role’s r4 instances is not possible.
Respective well-formedness rule is formed as follows [3]:

Rule 6: An exclusion constraint cannot be specified between two roles attached to object-types one of which is specified as a subtype of the other.

In Figure 9 related metamodel fragment is presented. Subtype constraint is externalized through SubtypeConnection. The object type associates with subtype connection over specialization and generalization associations depending on the place token in subtype constraint (either subtype or supertype).

Corresponding to the Rule 6 OCL expression is formed as:

```ocl
class Constraint

inv contradictionOfExclusionAndSubtype:
  self.oclAsType(SetConstraint).setType = SetConstraintKind::exclusion implies
  self.oclAsType(RoleConstraint).theRoleReference.role->forAll(r1:Role,r2:Role|r1<>r2 and r1.objectType.generalization->notEmpty() implies
    r1.objectType.generalization.superType ->excludesAll(r2.objectType.specialization.subType)
)
```

The invariant above indicates that if role’s instance is restricted by exclusion constraint, then no subtype connection possible between pair of object type instances related with role’s instances restricted by exclusion constraint.

Hence, in this section we had presented a subset of ORM well-formedness rules expressed with the OCL within MOF metamodel for ORM. Presented rules should be refined and expanded to be able to validate more complicated schemas than it was presented. New rules to capture more well-formedness aspects should be introduced as well. But it is for the further research. In the next section we will discuss on implementation of ORM well-formedness rules in MDA based modelling tool.

### 3 Implementation of ORM Well-Formedness Rules

#### 3.1 The Architecture of Prototype Tool

The prototype tool for implementation of ORM well-formedness rules is a part of BRidgeIT tool, a system for specifying business rules using templates and transforming them to other forms [6][7][8]. It has been implemented using Model Development Tool (MDT) and Eclipse Modelling Framework (EMF) plug-ins for Eclipse integrated development environment. We chose EMF, because it doesn't require a completely different
methodology or any sophisticated modelling tools [20]; moreover, it is MDA compliant [3]. Therefore, many relative projects are developing on this framework. Combination of EMF with MDT (Figure 10) provides possibility for us to specify OCL constraints within MOF metamodel for ORM and to create ORM model editor plug-in from it.

**Figure 10. The main components of prototype tool**

The architecture of prototype tool is provided by EMF, and the core of the prototype tool is graphically presented in Figure 11. Most of the data needed by the EMF generator is stored in the core model: ORM metamodel in *Ecore* format with attached OCL constraints. *Ecore* is meta-metamodel aligned on MOF [18]. A generator model (GENMODEL) is generated from ORM metamodel. It provides access to generated code repository, used prefix for generated factory and package class names, including the *Ecore* part, by wrapping the corresponding core model. The templates package consists of templates employed to specify references of OCL expressions used in static fields, operations and references of expressions with validation mechanism. ORM>Edit provides content and label provider classes, property source support, and other convenience classes that allow ORM models to be displayed using standard EMF viewers and property sheets. Moreover, it provides the command framework, including a set of generic command implementation classes for building editors. ORM.Editor is functional editor plug-in that allows to view instances of the model using several common viewers and to make common operations on model objects, or to modify the object properties in a standard property sheet.

**Figure 11. Architecture of prototype tool**

### 3.2 Implementation of the rules

Implementation of well-formedness rules of ORM in prototype tool includes the following steps: specification of OCL expressions, assignment of rules to *Ecore* metamodel, generation of model editor plug-in and verification of implemented rules with generated plug-in. Rules expressed in OCL are linked with metamodel element over annotation (*EAnnotation*) elements (Figure 12). Two annotation elements are used to specify constraints in the context of model element.

The first, named *Ecore* is used to define applied constraints by providing their names and the second one, named OCL, is used to preserve OCL expression within metamodel element. As it shown in Figure 12, OCL expressions are denoted as annotation detail element value.
In the generator model created from core model, plug-in variables and templates repository are defined. Model plug-in variable links generator model with OCL plug-in, used in this prototype to enable validation of OCL constraints. Generated model editor includes validation class, generated from templates associated with metamodel. As it was mentioned before, it also provides functionality to manage related models based on core metamodel.

3.3 Validation of the rules

A number of ORM schemas regarding to previously discussed rules as well more complex schemas were tested to check adaptability of the proposed approach. Figure 13 presents validation results of Rule 1 (described in subsection 2.1) where sample model violating this rule was validated. The dialog indicating validation problems enumerates the list of violated constraint during validation of ORM model.

A more complex model capturing various ORM constraints was formed according to ORM approach presented in [11]. As an example, we consider a fragment of university domain ORM model (Figure 15). It is fragment of information system used for registration of students.
The ORM model was used to check the correctness of implemented rules. The source ORM model was encoded to XMI format according to ORM metamodel and validated with generated tool. A validation result presented in Figure 15 shows that well-formedness rules implemented in modelling tool react only on those cases they are defined for. Therefore, succeeded experiment brings new opportunities to make a further research on this topic.

Currently, the tool we are developing supports validation of ORM well-formedness rules described in previous subsections. Additionally, we have implemented validation of a subset of fundamental constrains, such as rules for naming of model elements, compatibility of bounds used for ranged values as well as possible contradictions between set comparison constraints, and rules for compatibility of ring constraints.

4 Conclusions and Further Research

In this paper we presented an MDA based approach for specification and implementation of ORM well-formedness rules. A subset of known ORM well-formedness rules were specified using OCL in context of MOF metamodel for ORM. Implementation of metamodel in MDA based prototype tool was provided as well.

The OCL expressions for ORM well-formedness rules presented in this paper could be refined and adopted to include more than only those trivial cases. Moreover, it could be discussed on their adoption to concrete cases of ORM schemas. However, specification and implementation in prototype tool as well as successful test cases let us believe that presented approach could be expansive for further research.

Implementation of rules provides possibility to detect additional rules for ORM model to be well-formed and to refine them by checking with appropriative model schemas. Selected development environment is powerful enough to be able to generate modelling tool and provides all required APIs for further development. Therefore, there are several main further directions in tool development process: investigation on all well-formedness rules of ORM, development of graphical user interface to support ORM graphical notation. In
addition, verbalization mechanism as well as possibility to convert models to other MDA compliant modelling approaches and generation of executable code could be implemented.

5 References


Abstract. Constraints’ modeling is an important part of ontology engineering process. Today there are a lot of tools and languages provided for building constraints within different ontology development platforms. The choice of language usually depends on the developer’s opinion and experience, but sometimes the designed domain puts rigid restrictions on the language. In this research authors compare several constraints languages and analyze the best option for the ontology that is used to solve the challenges of integrating SAP CRM system.

Keywords: constraints modeling languages, Protégé editor, Enterprise Resource Planning systems implementation.

1 Introduction

In recent years ontologies have become a widespread technique for knowledge representation and knowledge management. A lot of researches are devoted to the problems of building ontologies in different areas like medicine, digital libraries, software engineering, system configuration, natural language processing, database design, Web search, etc.

The growing popularity of ontologies also stimulates the development of various tools and languages that can be used within ontology engineering process. The correct choice of appropriate technology is one of key factors to the successful ontology implementation, but sometimes it is also a challenging task for the development team. Usually the choice of technology is driven by selection of an ontology implementation language, and is based on the knowledge representation capabilities and inference mechanism needed by the application that uses ontology. Analyzing correspondence between the application requirements and the ontology language it is necessary to find out the answers to the really complex questions like:

- Is the expressiveness of ontology language deep enough to build the application domain?
- Does the inference mechanisms attached to an ontology language support the reasoning queries that application users can ask?
- Does any ontology development tool support the language?

Among such question one of the most difficult is the analysis of techniques or language forms that can be used to specify constraints over the concepts and relations within the ontology. The reason for this challenge is that it is very difficult to preview all necessary constraints in the beginning of the ontology modeling. Besides very often the ontology development tools provide additional languages or technologies to build constraints or rules for inference, and that’s why it is not enough to review ontology modeling language capabilities but it is also necessary to investigate the tools features. So, the problem of selecting a constraint language for a specific ontology can be viewed as a separate and pretty complex task of ontology development process.

Within this article we would like to present the results of comparative analysis of some constraints languages for the ontology that is used to support implementation process of solutions on the base of SAP Customer Relationship Management (CRM) system. Within the first part of the work we explain the purpose of this ontology and the basic requirements to the constraints modeling language that are set by the domain. The second section of the work is devoted to the scope and methodology of research that was used for comparison. And in the third section we provide the description of the languages and the results of comparison.

2 Designing Ontology for the Task of Integrating ERP Systems

Information systems play an expanding role in all aspects of business. More and more organisations use the information systems to support the key business processes. Within this tendency the number of companies that implement Enterprise Resource Planning (ERP) solution also grows. Here by ERP we mean not only the systems that functionality corresponds to ERP standards, but also any information system that helps to manage key areas of the business from the beginning to end, e.g. customer relationship management systems, supply chain management systems, strategic planning systems, etc.
But unfortunately despite the experience growth and development of many integration methodologies, implementation of ERP systems is still a risky step for any organisation. The statistic shows that risks of failure for ERP integration project are pretty high. The survey of the Standish Group, conducted in 2005, demonstrated that that 15 percent of such projects fail (never finish), 51 percent are challenged (cost, schedule or scope goals are not met), and only 34 percent are successful (meet all three criteria) [13].

Many researches are conducted to investigate the reasons of the faults and key success factors. In summary, the authors agree that there are three main problems: people, organisational issues and change management [1, 2, 3, 7, 8, 9]. The most common challenges here are end-user not being ready, resistance to change, lack of communication and support documentation, lack of resources, lack of commitment from top leadership and technical infrastructure problems [11].

The specific problems of ERP implementation projects seem to be very difficult and practically non-resolvable. But if we analyze these challenges from developers’ or integrators’ view point, we can find simple and logical reasons for them.

First of all, current ERP systems provide a numerous number of functional scenarios and constantly enlarge the volume of functionality. For example, within Customer Relationship Management (CRM) system provided by SAP AG Corporation, the number of business scenarios is more than 300, and overview training about this product takes 4 – 6 weeks.

But despite the growing volume of ERP functionality, many organisations still can’t find corresponding processes. The survey described in [8] showed that about 44% of respondents found organisation specific incompatibilities where the software did not support the way the organisation worked. For example, the software did not support many legislated procedures required by the public sector organisations. About 37% of the respondents reported few business specific incompatibilities. For example an organisation in the semiconductor industry found that the systems they had implemented did not support yield management requirements. About 37% found country specific incompatibilities. For example, the system did not support the bilingual requirements (French support was lacking for some applications) posed by some departments [8].

To make ERP solutions more flexible, vendors provide different ways of systems configuration and modification. On the one hand, it gives a very useful opportunity to the customers to adjust system functionality to the user requirements. But on the other hand, it makes the process of implementation more complex because very often to enable some functionality (like orders search or campaign management) it is necessary to pass through several configuration scenarios, that are interconnected and data dependent. One of the brightest examples here is security management. This area usually requires a lot of efforts from project team to complete the analysis and understand user groups, their roles, access level required by each group to the specific data, etc.

But ERP configuration approach can make this process even more difficult. For example in SAP R/3, hundreds of authorization objects are used to allow access to various actions in the system. A small or medium sized organisation may have 100 transactions that are commonly used, and each transaction typically requires at least two authorisation objects. If the company has 200 end users who fill a total of 20 different roles and responsibilities, there are approximately 800,000 (100^2*20*200) ways to configure security in the ERP system—and this scenario excludes other complexity factors, such as multiple transactions sharing the same authorisation objects, an authorisation object having up to 10 fields that can be assigned various values, and the possibility of using position-based security

But in spite of a variety of configuration methods that are provided within modern ERP, many companies still have to do system modifications. It is a third reason for many ERP specific challenges. Usually by modification we mean some changes to the system logic, user interface or database that are done though coding, database re-design or integrating new system self-developed modules on the same platform. Even nowadays modifications are still required for many companies. The survey [8] showed that about 65% organisations had to make some system modifications to meet end user requirements and about 50% of companies developed their own add-ons.

The problem of modifications is that ERP suppliers do not guarantee that upgrade packages won’t destroy customer-developed functionality. The upgrade package can change business process logic, database scheme, programs signature or even whole user interface screen. Appliance of such upgrade packages is not mandatory but sometimes ERP vendors do not support system of very old versions and require system upgrades. In this case the only thing that can be done on customer side is a detailed regression testing that is very time consuming. The examples of such upgrade package that can be mentioned here are Support Packs that are provided by SAP for all their systems. Support Packs usually contain fixes for the defects that were found by different customers within last 3-4 months, so that the scope of changes is not pretty big. But if within your system 10% of business processes are affected by system modifications, regression testing can require 7 – 8 weeks of testing with team up to 40 people.

Another problem of modifications is that support services of big ERP vendors usually do not provide help if there are any problems with your custom developed programs but not with native functionality. Customer
is not able to rely upon vendor’s support and needs to have his own support team who can manage modifications and provide help if necessary.

From the authors’ opinion, these challenges are the key factors of the ERP implementation projects failures. They make the process of ERP integration more and more difficult and put new questions and tasks before a project team. The project team needs not only to satisfy user requirements by developing the corresponding system features but also to answer a group of specific ERP implementation questions:

- How is it possible to use native ‘Off The Shelf’ (OTS) provided functionality as much as possible to build solution that will fully satisfy customer’s requirements?
- If it is not possible to use OTS, what ways of configuration can be used instead of modifications?

These questions are not difficult from the first point of view but that is a serious problem in each particular practical cases. The main challenge here is to collect people with all required knowledge to solve the technical issue and investigate it from different prospective.

The traditional method of resolving this problem is to form a team that will contain as many people as possible with knowledge of different components. But even high professional and experienced teams usually need to revert back to the training materials, help books, system help, use scenarios and other types of document that describe system business processes and configuration approaches. The problem here is that documentation analysis is usually time consuming activity because these materials contain some unstructured and not formalized information.

From the authors’ prospective there is another solution to the problem of knowledge collection. We consider that it is necessary not only to build the team of experienced specialists but also provide them with knowledge management or decision support system that can help to retrieve required information more quickly than manual document analysis and to find most effective configuration approaches. From functionality prospective the proposed system should:

- Contain structured and formal description of ERP functionality
- Contain structured and formal description of configuration methods and scenarios
- Be able to propose configuration according to customer’s requirements

To satisfy these requirements we propose to use ontologies as a method to collect and structure information. To illustrate how ontology can be used within ERP implementation authors developed such system for Customer Relationship Management (CRM) solution provided by SAP Corporation.

SAP CRM system provides users different functionality related to the business scenarios that happen between the company and its customers. As the volume of SAP CRM functionality is pretty huge, within the first attempt to build ontology of this system we decided to limit the scope of ontology by the main processes of managing master data objects within sales scenario: Account & Contacts Management, Organisational Management and Territory Management.

From general prospective our ontology consists from four sub ontologies:

- ontology of requirements,
- ontology of main master data objects,
- ontology of business processes and
- ontology of configuration objects

Within this article we are not going to describe all these ontologies in details, but it is necessary to underline the main constraints that are required in this ontology. So, we used constrains to define three types of limitations:

- Limitation of attributes definition: when the possible attribute values can be selected only from the list that is defined by particular configuration object
- Limitation of configuration objects usage: when it is necessary to follow special configuration scenario because settings of one configuration object are dependant on the settings of another one.

Besides as the main functionality of our system should be to find out necessary configuration to satisfy customer’s requirements, it can be great if the system can determine the missing configuration objects and create them automatically.
3 The Scope and Methodology of Research

3.1 The scope of the research

The scope of our analysis is limited to the comparison of constraint languages and some other tools that are available with the special ontology modeling environment Protégé. We have chosen this platform for ontology engineering because it is a free open source product with a lot of available plugins.

Today Protégé is one of the most popular open source editors for ontology engineering. The line of Protégé tools is developed by the Stanford Medical Informatics (SMI) group at Stanford University. The first Protégé editor was created in 1987; its main aim was to simplify the knowledge acquisition process for expert systems. To achieve this objective, it used the knowledge acquired in previous stages of the process to generate customized forms for acquiring more knowledge. Since then, Protégé has gone through several releases and has focused on different aspects of knowledge acquisition (knowledge bases, problem solving methods, ontologies, etc.).

Protégé knowledge model is based on frames and first order logic. It is OKBC compatible, which means that the main modeling components of Protégé are classes, slots, facets and instances. Classes are organized in class hierarchies where multiple inheritance is permitted and slots can also be organized in slot hierarchies. Classes in Protégé can be concrete or abstract. The former may have direct instances while the latter cannot have them; instances of the class must be defined as instances of any of its subclasses in the class taxonomy. Slots are global to the ontology (two different slots cannot have the same name in an ontology) and can be constrained in the classes to which they are attached.

Like in OKBC, Protégé also distinguishes between two types of slots: template and own slots. The standard facets for template slots in Protégé are: NL documentation, allowed values, minimum and maximum cardinality, default values, inverse slot and template-slot values. But if necessary, other facets can be created and attached to the slots to describe them.

3.2 The Methodology of the Research

From the prospective of the modeling ontology, it was decided to compare constraints languages by their abilities to:

- Build common constraints like:
  - Cardinality constraint constrains how many values are allowed for a slot on a frame. This may be a lower bound, an upper bound or an interval.
  - Concrete value constraint constrain the value of a slot, either by defining a list of all the allowed values (implicitly disallowing all other values), or by listing all disallowed values (implicitly allowing all other values).
  - Value range constraint constrains the allowed range of values for a numeric slot. This may be a lower bound, an upper bound or an interval.
  - Value class constraint constrains the value of a slot in a wider way than the two above. This constraint states that the value of a slot must be a subclass or an instance of another frame.

- Be enough expressive to build all described limitations
  - Limitations of attributes definition
  - Limitations of configuration objects usage
  - Limitations of object attribute availability

Also within the languages analyzes the following factors were taken in the account:

- The difficulty of language syntax
- The possibility to create additional individuals in the ontology

4 Comparative Studying of Some Constraint Languages in Protégé

4.1 Protégé Axiom Language (PAL)

We have started our research from analyzing the standard constraint language of Protégé – Protégé Axiom Language. The Protégé Axiom Language (PAL) extends the Protégé knowledge modeling environment with support for writing and storing logical constraints and queries about frames in a knowledge base. More than just a language, PAL is a plugin toolset that comprises engines for checking constraints and running queries on
knowledge bases, as well as a set of useful user interface components. The primary use of PAL is to detect incomplete and/or inconsistent entry of information in a Protégé knowledge base.

When entering instances in Protégé, an end-user is guided by facet-based constraints that impose restrictions on the values that can fill each slot. However, these facet-based constraints only ensure that the local value of each slot is valid; they do not monitor the relationships among values of different slots of an instance, nor of different instances. The Protégé Axiom Language provides a way to make arbitrary assertions about groups of concepts. The PAL engine can check to make sure that the assertions hold across the entire knowledge-base. For example, if, in the newspaper project, we know that editors always earn more than the employees they are responsible for, we can express that piece of knowledge as a constraint. Then, checking the constraint tells us if certain instances of employee or editor have been entered incorrectly in the knowledge base. Similarly, queries can be created to determine which instances satisfy a particular assertion.

Although PAL is fundamentally a constraint- and query-creation language, the PAL plug-in provides much more:

- A language to express logical constraints and queries about frames in a knowledge base;
- A set of special-purpose frames to model constraints and queries as part of a knowledge base;
- A structured PAL Expression Editor that provides context-sensitive help to write PAL sentences;
- A constraint-checking engine, which can be invoked either with the PAL Constraints Tab or programmatically;
- A querying engine which can be invoked either with the PAL Queries Tab or programmatically.

PAL constraints have almost the expressibility of first order predicate calculus. This provides the ability to state arbitrary and complex constraints. Analysis of PAL language has shown that practically all types of constraints are possible to build:

- Cardinality constraint: To specify cardinality constraint it is possible to use the properties of the slot or specify a PAL constraint. For example the constraint that “For all articles, the number of values for the slot keywords should be greater than 2”, can be expressed by the following way:

  (defrange ?article :FRAME Article)
  (forall ?article
    (> (number-of-slot-values keywords ?article) 2))

- Concrete value constraint: It is possible to define a list of allowed values for a slot in Protégé by default. But to define disallowed values you have to write a PAL constraint.

- Value range constraint: Protégé slots have two facets named minimum and maximum that provides the ability to define the range of numerical values. If you need to specify the value range constraint for any other type of slot, you have to write pretty complex constraint and list in it all possible values.

- Value class constraint: Can be built easily in PAL through the logical expressions. But if the limitation of available classes is really strong and is dependent on the particular slots properties, it can be impossible to formulate such constraint.

From the first point of view PAL looks like a perfect solution, but within our research we have found that we are not able to formulate all necessary constraints in this language. So, we were able to create the constraints to limit the possible values of attributes that are defined through configuration objects. As an example we can demonstrate the structure of the following constraint: “Business partner can have a specific role only if it is defined in a special configuration object”. PAL expression here will be the following:

  (defrange ?businessPartner :FRAME MD_Business_Partner)
  (defrange ?roles :FRAME Define_BP_Roles)
  (forall ?businessPartner
    (exists ?roles
      (= ('BP_Role' ?businessPartner) ('BP_Role_Id' ?roles))))

Another type of constraint that we were able to specify in PAL was a limitation of configuration objects usage. As an example here we can take a limitation that “Depending on the group, business partner should have unique id from a specific value range. The dependency between value range and group is defined separately in the configuration object”. The constraint for this example is expressed by the following way:

  (defrange ?businessPartner :FRAME MD_Business_Partner)
  (defrange ?groupingAndRating :FRAME Define_Groupings_and_Assign_Number_Ranges)
  (defrange ?range :FRAME Define_Number_Ranges)
forall ?businessPartner
(exists ?groupingAndRating
(exists ?range
(= ('BP_Grouping' ?businessPartner) ('BP_GR_grouping' ?groupingAndRating))
(= ('BP_GR_number_range' ?groupingAndRating) ('BP_NR_no' ?range))
(> ('BP_Identification_Number' ?businessPartner) ('BP_NR_From_Number' ?range))
(< ('BP_Identification_Number' ?businessPartner) ('BP_NR_To_Number' ?range)))))))

But unfortunately we were not able to find a suitable constraint expression that can be used to formulate the dependencies between configuration settings and business object attributes availability. The problem here was that we could not specify in advance what particular value of the controlling object will be used to limit the available attributes and that’s why we could not specify the condition statement of constraint.

Also during the work we found several more disadvantages of PAL language:

• The syntax of this language is pretty complex and difficult to understand. It’s really hard to formulate assertions in PAL logic. Even beyond getting the syntax right, it’s very hard to be certain you’ve expressed the underlying assertion correctly. And assertions are hard to read as well.

• There is no way to modify the ontology during the evaluation of constraints or propose the user how to resolve conflicts. The PAL Protégé plugin just select the individuals that do not correspond to the constraints and show them separately. Sometime the ways to fix the discrepancy in the individuals is really easy but it is not possible to build such rules into the system using PAL.

As far as we were not able to build all necessary constraints and provide a method how to fix ontology discrepancies by using PAL language, we started to investigate other languages and tools to achieve our aim. The next option that we analyzed is Algernon tool.

4.2 Algernon

Algernon is not a constraint language directly; by the fact it is an inference engine that allows formulating and executing special rules. But from Protégé prospective it is one of additional plug-ins that provide an ability to express the constraints over the ontology and that’s why we also analyzed this tool in our research.

Algernon is an inference engine that supports both forward and backward chaining rules. It automatically activates forward chaining rules when new data is stored in the ontology and it automatically activates backward chaining rules when the knowledge base is queried. If a forward chaining rule contains a step that queries the knowledge base, Algernon will automatically switch to backward chaining mode in order to satisfy the query using any applicable backward chaining rules.

Algernon guarantees that rules will not be activated more than once for each identical assertion or query. The first time a query is made, backward chaining rules will be used to infer the answer. If later an identical query is processed, the result will be looked up in the knowledge base rather than wasting time re-executing rules to infer knowledge that is already in the knowledge base. Algernon correctly handles special cases such as new rules being added between queries, and some situations where frames are deleted after a query is made. However, it does not have full support for non-monotonic reasoning.

From constraint building prospective we were able to specify all required limitations in Algernon. From the general prospective all constraints can be built using this engine:

• Cardinality constraint: To specify cardinality constraint it is possible to use the properties of the slot or specify a COUNT command of Algernon.

• Concrete value constraint: It is possible to define a list of allowed values for a slot in Protégé by default and use some simple Angernon rules for that. But it is not possible to determine not allowed values as there is no negation in the rules statements in the Algernon.

• Value range constraint: Is very difficult to express as there is no ‘at least’ or ‘at most’ commands. It is possible only to count number of required slots and process that information within backward rules

• Value class constraint: It is possible to specify a wide and deep class of constraints for limiting the slots classes values, but all of them should be expressed if the form of if – then clauses and are not linked to the classes directly.

The biggest advantage of Algernon tool for our task is definitely the possibility to modify the ontology during rules processing. This engine provides a set of commands that allows creating and deleting ontology classes, relations, individuals and facets within checking the correspondence of instances to the rules. But
Unfortunately, processing the rules Algernon gives either the list of individuals that satisfy the constraint or just gives a statement that rule execution is not possible without giving the items that can’t be processed.

Given this result, we came to the conclusion that this tool also can’t be used in ERP ontology but rule processing functionality is a possible way to provide a user with a list of entries that must be created within a system. Trying to find a technology which combines all the constraint building features and provides a possibility to work with rules engines, we moved to the option of using OWL language and SWRL rules.

4.3 OWL Language and SWRL Rules

The Web Ontology Language (OWL) is the most recent development in standard ontology languages, endorsed by the World Wide Web Consortium (W3C) to promote the Semantic Web vision. Protege allows to create ontologies in OWL and provide a special version of the editor that is called Protege-OWL.

OWL ontologies have similar components to Protégé frame based ontologies. However, the terminology used to describe these components is slightly different from that used in Protégé. An OWL ontology consists of individuals, properties, and classes, which roughly correspond to Protégé instances, slots, and classes.

Individuals are very similar to instances; they represent objects in the domain that we are interested in. An important difference between Protégé and OWL is that OWL does not use the Unique Name Assumption (UNA). This means that two different names could actually refer to the same individual. For example, “Queen Elizabeth”, “The Queen” and “Elizabeth Windsor” might all refer to the same individual. In OWL, it must be explicitly stated that individuals are the same as each other, or different to each other — otherwise they might be the same as each other, or they might be different to each other.

Big difference between Protégé notation and OWL language is the replacement of slots by properties. Properties are binary relations on individuals — i.e. properties link two individuals together. Properties can have inverses. For example, the inverse of hasOwner is isOwnedBy. Properties can be limited to having a single value — i.e. to being functional. They can also be either transitive or symmetric.

But the biggest difference of OWL from the prospective of our research is that this language provides a wide range of constructions to build constraints over the classes. As OWL is built on the descriptive logic concept, it provide following constructs to build the logical constraints:

- intersectionOf
- unionOf
- complementOf
- oneOf
- toClass
- hasClass
- hasValue
- maxCardinalityQ
- minCardinalityQ
- cardinalityQ

So with these constructors we were able to specify all necessary constraints that characterize the classes. But to build the inter-classes constraints and to be able to manipulate the individuals on the base of logical constraints, we found additional tool that provides such functionality — SWRL rules.

SWRL is an acronym for Semantic Web Rule Language. It is intended to be the rule language of the Semantic Web. SWRL is based on the OWL Web Ontology Language. It allows users to write rules to reason about OWL individuals and to infer new knowledge about those individuals.

A SWRL rule contains an antecedent part, which is referred to as the body, and a consequent part, which is referred to as the head. Both the body and head consist of positive conjunctions of atoms. SWRL does not support negated atoms or disjunction. Informally, a SWRL rule may be read as meaning that if all the atoms in the antecedent are true, then the consequent must also be true.

SWRL rules are written in terms of OWL classes, properties and individuals. For example, a SWRL rule expressing that a person with a male sibling has a brother would require capturing the concepts of person, female, sibling of and brother of in OWL. Intuitively, the concept of person and male can be captured using an OWL class called Person with a subclass Man; the sibling and brother relationships can be expressed using OWL object properties hasSibling and hasBrother with a domain and range of Person. The rule in SWRL would then be: Person(?p) ^ hasSibling(?p,?s) ^ Man(?s) -> hasBrother(?p,?s)

Within Protege OWL editor there is a special plugin that allows to build SWRL rules in a graphical form, save them, import to the inference engine, execute the reasoning mechanism and see the changes or discrepancies that were found by the system. Such approach seems very reasonable with the task of our ontology...
especially as SWRL allows to create some instances within the inference process and user can decide if he wants to put the proposed items into ontology or not.

5 Conclusions

So as a result of our research we found that the most suitable constraint modeling approach for the task of building ERP ontology is the using OWL language with SWRL rule engine. This approach gives the ability to specify all types of necessary constraint like

- Cardinality constraint
- Concrete value constraint
- Value range constraint
- Value class constraint

The OWL language with SWRL is also suitable to build required limitations for

- Attributes definition
- Configuration objects usage
- Object attribute availability

References


BUILDING ONTOLOGIES FROM MULTIPLE INFORMATION SOURCES

Raji Ghawi, Nadine Cullot

Laboratory LE2I, University of Burgundy, 21000 Dijon, France, [first.last]@u-bourgogne.fr

Abstract. Ontologies provide a promised technology to solve the semantic heterogeneity problem. In literature, many works tackle the development of ontologies from scratch or from one information source. But building an ontology from several heterogeneous information sources has not been well investigated by researchers. We propose an ontology-evolution-based approach to create an ontology from several heterogeneous information sources, where an initial ontology is evolved progressively by involving an information source in each step. Two types of information sources are considered, relational databases and XML documents. An evolution step includes a set of ontology change operations as well as a set of mapping bridge definitions between the entities of the ontology and the entities of the source being involved. The process of involving an information source in the development of an ontology is semi-automatic, that is, a human intervention is needed to suggest initial correspondences between ontology and information source entities. These correspondences are treated automatically to apply needed changes on the ontology and to elaborate suitable mappings. The automatic treatment of initial correspondences is based on pre-defined rules that determine which are the possible solutions for each possible case of correspondence.

Keywords: Building ontologies, Ontology evolution, Databases, XML.

1 Introduction and Motivation

In order to achieve an efficient integration of distributed heterogeneous information sources, conflicts of these heterogeneous sources have to be solved at syntactic, structural and semantic levels. In particular, semantic interoperability of information sources became critical issue. Ontologies provide a promised technology to solve the semantic heterogeneity problem, because they allow the representation of common semantics of the domain of discourse. Ontology has become one of the hottest issues of research among different communities (Artificial Intelligence, Databases, Semantic Web, etc.). An Ontology is defined as an "explicit specification of a conceptualization" [11], therefore, it can be used to describe the semantics of the information sources and to make their content explicit. In ontology-based information integration approaches, ontologies are mainly used for the explicit description of the information source semantics. According to the way of how ontologies are employed in information integration, three different approaches can be distinguished: single ontology approach, multiple ontologies approach and hybrid approach. Single ontology approach uses only one global ontology. All information sources are linked to this global ontology by relations expressed via mappings that identify the correspondence between each information source and the ontology. In multiple ontologies approach, each information source is described by its own ontology and inter-ontology mappings are used to express the relationships between the ontologies. The hybrid approaches combine the two previous approaches. Each information source has its own ontology and the semantics of the domain of interest as a whole is described by a global reference ontology. In these approaches there are two types of mappings: mappings between an information source and its local ontology and mappings between local ontologies and the global ontology.

In addition to content explication, ontologies can also be used as means to interrogate information integration systems. That is, the user formulates a query in terms of the ontology, then the integration system decomposes the global query into sub-queries for each appropriate source, collects and combines the query results, and returns the results.

Whatever the role of ontologies, the important question that arises is how to obtain them? The development of ontologies has been investigated by many researchers. In the literature, two major approaches of ontology development have been addressed, 1) ontology construction from scratch, and 2) ontology development using existing resources. It became clear that developing ontologies from scratch is costly and difficult [2,18]. Therefore, many development methodologies (such as [17,6]) include a phase of integration of existing ontologies, in which reusable components are imported as much as possible.

However, in the context of semantic interoperability, where ontologies are used to represent the semantics of underlying data sources (that may be relational databases or XML documents), it is crucial to involve the underlying information sources in the process of ontology development. In the literature, there are many approaches that handle the creation of an ontology from one existing information source (see section 2).

It is often necessary to involve more than one information source to build an ontology. We think that the creation of an ontology from several heterogeneous information sources has not been yet well investigated by
researchers. To our knowledge, there is no “suitable” approach to handle such process. The aim of this paper is to propose an integrated approach to build an ontology from several heterogeneous information sources.

During the creation of an ontology from information sources, an important point has to be taken in account which is the mappings between the created ontology and the underlying sources. The operations to elaborate and update these mappings have to be specified and handled. An appropriate mapping specification language is also needed to describe the elaborated mappings.

In this paper, we propose an ontology-evolution-based method to build an ontology from several heterogeneous information sources. This method reuses several techniques from various fields: database-to-ontology mapping, XML-to-ontology mapping, and ontology change management. Because of the lack of space, we only give a background on the fields of database-to-ontology mapping and ontology change management. In section 2, we give an overview of our proposed approach, whereas in section 4, we present some details on involving a database in our ontology-evolution-based approach to build ontologies. Section 5 concludes the paper.

2 Background

Ontology development often requires involving information sources. In particular, relational database and XML documents gain significant importance as dominant means to store information. On the one hand, databases are widely recognized as the backbone of information systems. Specifically, relational databases are the most widely used due to their simplicity and the mathematical formal theories that support them. On the other hand, XML brought interoperability at a syntactic level, and has reached a wide acceptance as data exchange format. In the following, we review some existing approaches of developing ontologies from relational databases.

2.1 Database-to-Ontology Mapping

The database-to-ontology mapping approaches can be classified into two main categories, 1) approaches that create an ontology from a database, and 2) approaches that map a database to an existing ontology.

2.1.1 Ontology creation from a database

The objective of these approaches is the creation of an ontology from a relational database and the migration of the contents of the database to the generated ontology. The mappings here are simply the correspondences between each created ontology component (concept, property, etc.) and its original database component (table, column, etc.). In these approaches, the database model and the generated ontology are very similar. Mappings are quite direct and complex mapping situations do not usually appear. In some approaches, the ontology creation process is straightforward, i.e., direct transformations of database tables and columns into ontology concepts and properties respectively. In other approaches, the creation involves the discovery of additional semantic relations between database components (such as the referential constraints) and takes them into account while constructing ontology concepts and properties.

Stojanovic et al. in [16] propose an approach to generate ontologies from relational databases based on pre-defined mapping rules. This approach takes in account different types of relationship between database tables, where mapping rules are defined for different situations. When several rules can be applied, a user intervention is needed to choose the most suitable rule.

DataGenie [9] is a tool that allows the automatic generation of an ontology from a relational database. This generation process is simple and direct, tables are transformed to classes, columns are transformed to properties, and foreign keys are transformed to object properties.

Relational.OWL [5] is an OWL ontology representing abstract components of a relational schema. The schema of any relational database can be described as an instance of this ontology and then be used to represent the data stored in that specific database.

2.1.2 Mapping a database to an existing ontology

In these approaches, the goal is to establish mappings between an existing ontology and a database, and/or populate the ontology with the database contents. In this case, the mappings are usually more complex than those in the previous category, because the modeling criteria used for designing databases are different from those used for designing ontology models [1], thus different levels of overlap (not always coincidence) can be found between the database domain and the ontology one.
KAON Reverse\footnote{http://kaon.semanticweb.org/alphaworld/reverse/view} is a tool for mapping relational database content to ontologies. Firstly, a set of mapping rules are defined manually to describe the relation between the database schema and the ontology structure, then according to these, the instances are automatically exported to the ontology. There are two principal types of mappings: 1) Table Mapping that relates a table to a concept, and 2) Column Mapping that relates a table column to an attribute or to a relation. A column mapping can only be defined in the context of a table mapping.

Vis-A-Vis \cite{8} is a tool that allows the mapping of relational databases to existing ontologies. A mapping is done by adding a new property to an ontology class. This property consists of an SQL expression which will be executed and return the dataset corresponding to the ontology class. This tool also performs a set of consistency checks to insure the validation of mappings. For example, two disjoint classes cannot have mappings to two datasets having common records.

2.1.3 Database-to-ontology mapping specification

Several languages have been proposed to formally express database to ontology mappings:

D2RQ \cite{3} is a RDF-based declarative language to describe mappings between relational database schemas and OWL/RDFS ontologies. The D2RQ Platform uses these mappings to enable applications to provide an RDF-view on a non-RDF database. In D2RQ, basic concept mappings are defined using class maps that assign ontology concepts to database sets. A class map specifies how instances of the class are identified, and it has a set of property bridges, which specify how the properties of an instance are created. Object property bridges should have a primitive that indicates the related concept bridge, and a join primitive that indicates how the related tables have to be joined together.

RZO \cite{1} is another declarative language that describes mappings between database schemas and ontologies. It is more expressive than D2RQ as it provides an extendable set of condition and transformation primitives. After the manual generation of a RZO document, it is processed by a generic query engine, called ODEMapster, that automatically populates the ontology with instances extracted from the database content. This operation can be done in two modes: 1) query driven, i.e. parsing a specific query and translating its result, or 2) massive dump, i.e. creating a semantic RDF repository and translating the full database to it.

2.2 Ontology Change Management

The problem of modifying an ontology in response to a certain need is known in the literature as the ontology change problem. Flouris et al. \cite{8} identify and distinguish several facets of this problem such as: ontology alignment, merging, mapping, evolution and versioning. In particular, they define the ontology evolution as "the process of modifying an ontology in response to a certain change in the domain or its conceptualization" \cite{7}.

When an ontology is evolved, it is very often required to determine whether the new version of the ontology is compatible with the old version. In databases, compatibility usually means the ability to access all of the old data through the new schema (the change does not cause any loss of instance data). Noy et al. distinguish in \cite{13} several dimensions of compatibility of ontologies, namely: 1) instance-data preservation, where no data is lost in transformation from the old version to the new one, 2) ontology preservation, where a query result obtained using the new version is a superset of the result of the same query obtained using the old version, and 3) consequence preservation, if an ontology is treated as a set of axioms, all the facts that could be inferred from the old version can still be inferred from the new version.

Noy et al. also distinguish two modes of ontology evolution: traced and untraced evolution \cite{13}. In the traced mode, the evolution is treated as a series of changes in the ontology. After each change operation, we consider the effects on the concerned parts (the instance data, related ontologies and applications) depending on the used dimension of compatibility. The traced mode is quite similar to schema-evolution.

Stojanovic et al. present in \cite{15} a complete ontology evolution process consisting of six phases. In the change capturing phase, the changes to be performed are identified. In the change representation phase, the changes are formally represented. In order to guarantee the validity of the ontology at the end of the change process, any problems that will be caused when the required changes are actually implemented have to be identified and addressed. This is the role of the semantics of change phase. In the change implementation phase, the changes are physically applied to the ontology. The change propagation phase follows, where the implemented changes need to be propagated to all concerned parts such as the instances, the dependent ontologies and applications. Finally, the ontology engineer can review the changes and possibly undo them during the change validation phase.

There are two major types of changes, 1) atomic (elementary) changes which are simple and fine-grained such as the addition of a concept, and 2) complex (composite) changes that are more coarse-grained. It is
possible to replace a complex change by a series of atomic changes, but it is not generally appropriate because this may cause undesirable side-effects [15].

Change operations can also be classified according to the change effects with respect to the instance-data–preservation dimension of compatibility [13], i.e. whether instance data can still be accessed through the changed ontology. Three operations effects are distinguished, 1) information-preserving changes, no instance data is lost, 2) translatable changes, no instance data is lost if a part of the data is translated into a new form, and 3) information-loss changes, it cannot be guaranteed that no instance data is lost. Usually, addition operations are information-preserving, removal operations are information-loss, merge and split operations are translatable changes.

3 Building Ontologies from Several Information Sources

Ontology building from several information sources is the process of creating a target ontology model by combining the components of several overlapped source models, and establishing semantic bridges between the components of the created ontology and their corresponding components of information sources. As we have seen in the previous section, there are several approaches that address the problem of building one ontology from one information source (a database or an XML document). However, to our knowledge, there is not yet any works in the literature that address the problem of creating an ontology from several heterogeneous information sources.

We think that two ways are intuitively possible to tackle this problem, 1) ontology merging based approach, and 2) schema merging based approach.

In ontology-merging-based approach, an ontology (called partial) is independently created from each information source using existing available tools. Then, all partial ontologies are merged consecutively for giving the final ontology. There are several approaches and tools for merging ontologies, we refer to [14] for a survey on this field. In this approach, mappings are elaborated between each source and the final ontology using existing mapping tools.

In schema-merging-based approach, the schemas of all information sources are merged consecutively, using existing schema-merging approaches, to give a unified schema. Then, the target ontology is created from this unified schema. The suitable mappings are elaborated between each source and the final ontology using existing mapping tools.

However, in this paper we propose an alternative approach to create an ontology from several heterogeneous information sources that is based on ontology evolution.

3.1 Ontology-Evolution-based Approach

In our ontology-evolution-based approach, the target ontology evolves each time a new source is added. The process starts with an initial ontology that can be an existing ‘autonomous’ ontology or created from the first source\(^2\). Then, each time a ‘new’ information source is involved, the current ontology is evolved by adding new concepts and properties from the involved source. We will use the term ‘evolution step’ to denote the evolution of the ontology from its old version to a new version when an information source is involved. An evolution step can be seen as the process of mapping of an information source to an existing ontology, but with a change of the ontology when necessary. That is, the following general strategy is used:

- If a source entity (database table or column, etc.) corresponds to an ontology entity, then we establish a mapping bridge between them, and we do not change the ontology.
- If a source entity has no corresponding ontology entity, then we change the ontology by creating an entity that corresponds to the source entity, and we establish a mapping bridge between the source entity and the ontology entity that we created.

Figure 1 illustrates the evolution step. Firstly, a human expert indicates initial correspondences between the entities of the information source being involved and the entities of the current ontology. Then, the initial correspondences are automatically treated according to pre-defined rules. The result of this treatment is a set of ontology change operations that are required to modify the old version of the ontology towards the new version, as well as, a set of mapping bridges between the entities of the ontology (the new version) and the entities of the information source.

The mapping bridges are saved to a mapping document that describes the relationship between the new version of the ontology and the current information source. Furthermore, it is necessary to review and update the mappings between the ontology and the previously involved sources in order to adjust these mappings according to any changes occurred in the ontology.

\(^2\) As mentioned in section 2.1.1, there are several approaches to create an ontology from a single database. However, we use our own tool, called DB2OWL, for this purpose [4,10].
In order to keep all the mapping documents (the current and the previous) consistent with the ontology, the ontology change operations are executed in a transactional mode. A transaction is composed of an ontology change operation, a set of related mapping bridge definitions and a set of depending previous-mappings updates. Thus, when a transaction is committed, the ontology change operation is executed, its related mapping bridges are saved to the mapping document, and its depending previous-mappings updates are applied on the concerned previous-mappings documents.

In our approach, the evolved ontology is expressed using OWL language, which is a W3C recommendation for publishing and sharing ontologies on the web. We use a mapping specification similar to D2RQ [3]. We extended it with additional primitives to express transformations, compositions and conditions. Mapping documents are written in XML, and they are used for query resolution purposes (this topic is beyond the scope of the paper).

In the following section, we present the set of ontology change operations that we use in our approach, as well as their effects on the ontology and on the previously established mappings.

3.2 Ontology Change Operations

In our approach to build ontologies by involving information sources, the target ontology is instance-free, that is, data instances still reside in their sources and are not exported to the target ontology. Instead, the relationships between the entities of the ontology and the entities of each information source are expressed via mapping bridges. Therefore, no ontology population by instances is performed, but a query-driven approach is used where data instances are retrieved from their sources only for query answering purposes using the elaborated mapping bridges.

Consequently, we replace the information-preservation dimension of compatibility by a similar dimension which is the mapping-preservation dimension of compatibility. We say that two versions of ontology are compatible if the mappings between the old version and information sources are still valid for the new version. According to this dimension, we distinguish three types of ontology change operations;

1. Mapping-preservation change operations; new mapping bridges may be added but existing bridges are not changed.
2. Translatable change operations; existing bridges are changed but not lost.
3. Mapping-loss change operations; existing bridges may be lost.

The ontology change operations that we will need in our approach to build ontologies by involving information sources are seven atomic operations and two complex ones. These few operations are sufficient to handle heterogeneity situations that may be encountered between the ontology and the information sources (see section 4). The atomic operations are the following:

- The addition of a new concept to the ontology (Add_Concept).
- The addition of a new datatype property to an existing concept (Add_DatatypeProperty).
- The addition of a new object property to an existing concept (Add_ObjectProperty).
- The setting of a concept as a subconcept of another one (Set_SubConceptOf).
- The setting of a property as a subproperty of another one (Set_SubPropertyOf).
- The setting of two ‘object’ properties as inverse of each another (Set_InverseOf).
- The removal of an existing property from the ontology (Remove_Property).
Except for the last one, all these operations are mapping-preservation, since they may add new mapping bridges but do not affect existing ones. The Remove Property operation is mapping-loss because it causes the loss of any mapping bridges concerning the removed property.

The complex operations are:

- The replacement of an existing datatype property by two (or more) ones (Split Property).
- The conversion of a datatype property into an object property (Convert Property).

These complex operations can be replaced by a series of atomic changes. The Split Property operation can be replaced by a set of Add DatatypeProperty and one Remove Property operations. The Convert Property operation can also be replaced by one Add ObjectProperty and one Remove Property operations.

However, since both replacements include the Remove Property operation which is mapping-loss, we do not do these replacements because both complex operations would become mapping-loss, too. Instead, we use an approach that let both Split Property and Convert Property become translatable change operations. In fact, mapping update transactions are embedded into the complex operations in order to modify the existing mapping bridges instead of remove them. The update of previous mappings is beyond the scope of this paper.

In the next section we present in details the process of involving a relational database in an ontology, as well as, the different cases that arise from the heterogeneity between them.

4 Ontology Evolution by Involving a Database

A relational database schema consists of a set of tables, each of them contains a set of columns. Tables may relate to each other using primary – foreign key relationships. The method we present here combines several techniques from several research fields:

1. Building an ontology from a database (DB2OWL [4,10])
2. Mapping an existing ontology to a database (D2RQ [3])
3. Ontology change management ([15])

In this section, we present a semi-automatic process to involve a database schema in an ontology. This process is supervised by a human expert, who firstly suggest initial correspondences between the ontology and the database entities. There are different cases of initial correspondences that can be suggested by the human expert. We defined a set of rules that associates each correspondence case with its possible solutions. Each solution consists of one or more ontology change operations and one or more mapping bridges.

The initial correspondences are treated automatically to apply needed changes on the ontology and to elaborate the suitable mappings. The automatic treatment of initial correspondences is based on pre-defined rules that determine which are the possible solutions for each correspondence case. Different possible solutions lead to different evolution strategies. Thus an intervention of the expert is needed again in order to validate one of the possible evolution strategies. The whole process consists of two main phases:

1. Evolving ontology concepts using database tables.
2. Evolving ontology properties using database columns.

4.1 Phase 1: Evolving Ontology Concepts Using Database Tables

In this phase, the ontology engineer identifies initial correspondences between the ontology concepts and the database tables. Then, these correspondences are automatically treated according to the rules presented below. The result of this treatment is a set of ontology change operations and a set of mapping bridges between ontology concepts and database tables. The validation of these operations and mappings is required to go on with the next phase.

In the following, we present some rules to solve different possible cases of correspondences between concepts and tables. For each case, the solution is some possible ontology change as well as some mapping bridges definitions that should be added to the mappings document. The different main cases that may be encountered are the following:

1. A concept C has no corresponding table T.
2. A concept C corresponds to exactly one table T.
3. A table T has no corresponding concept C.
4. A concept C corresponds to several tables T₁, T₂, …, Tₘ.
5. A table T corresponds to several concepts C₁, C₂, …, Cₙ.
6. Several concepts C₁, C₂, …, Cₙ correspond to several tables T₁, T₂, …, Tₘ.

Those cases are presented below with their solutions:
Case 1: A concept C has no correspondent table in the database. In this case, nothing should happen, the ontology does not change, and nothing is added to the mappings document.

Case 2: A concept C corresponds to exactly one table T. In this case, the ontology does not change, but a concept bridge between C and T is added to the mappings.

Case 3: A table T has no corresponding concept C. For this case, we will use the approach followed in our DB2OWL tool [4,10] where several cases of tables are distinguished: 1) a table is used to relate two other tables in many-to-many relationship, 2) a table is related to another table by a foreign key which is also a primary key (inheritance), and 3) other tables (default case). According to this distinction, we present three solutions for these three sub-cases (see Figure 2).

Figure 2. Phase 1: Case 3: A table T has no corresponding concept C

Case 3.1: A table T is used only to relate two other tables T1 and T2 in a many-to-many relationship. T1 and T2 are supposed to be already mapped to two concepts D1 and D2. In this case, no concept is created, but two inverse object properties between D1 and D2 are added to the ontology (see the case 3.1 in Figure 2).

Case 3.2: A table T is related to another table U by a foreign key which is also a primary key. In this case, a new concept C is added to the ontology and a concept bridge between T and C is added to the mappings. In addition, U is supposed to be already mapped to some concept D, therefore, C is set to be a sub-concept of D (see the case 3.2 in Figure 2).

Case 3.3: A table T is in the default case. A new concept C is added to the ontology and a concept bridge between T and C is added to the mappings (see the case 3.3 in Figure 2).

From an algorithmic point of view, the treatment should start by the default case (case 3.3) because it is needed for other cases, then the case of inheritance (case 3.2) is treated, and finally the case of many-to-many relationship (case 3.1). In case 3.2 and case 3.3, when a concept is created from a table, the suitable properties of the concept are created from the table columns in the phase 2.

Case 4: A concept C corresponds to several tables T1, T2, ..., Tm. According to previous cases, tables T1, T2, ..., Tm should be mapped to some (existing or created) concepts C1, C2, ..., Cm. However, two sub-cases are distinguished according to the type of correspondence: union and join (see Figure 3).

Case 4.1 (union): One concept C corresponds to the union of several tables T1, T2, ..., Tm. The solution consists in setting these concepts as sub-concepts of C. The common properties are moved to the super-concept C (see the case 4.1 in Figure 3).

Case 4.2 (join): One concept C corresponds to the join of several tables T1, T2, ..., Tm. In general, the join of tables does not correspond to a concept, except if the join is based on primary keys on both tables. The solution of this case consists in setting these concepts to be super-concepts of C (multiple inheritance) (see the case 4.2 in Figure 3).

Case 5: A table T corresponds to several concepts C1, C2, ..., Cn. In this case, a new concept C is added to the ontology and a concept bridge between T and C is added to the mappings. In addition, concerning concepts are set to be subconcepts of C, and common properties are moved to the super-concept C (see Figure 4).
Case 6: Several Concepts $C_1$, $C_2$, ..., $C_n$ correspond to several tables $T_1$, $T_2$, ..., $T_m$. Although this case could not happen in real world, but if it occurs, it can be decomposed into simpler cases. Either we consider that each concept $C_i$ corresponds to several tables $T_1$, $T_2$, ..., $T_m$, so we have $n$ occurrences of the case 4. Or we consider that each table $T_j$ corresponds to several concepts $C_1$, $C_2$, ..., $C_n$, so we have $m$ occurrences of the case 5.

At the end of this first phase, each table is mapped to one or more concepts (except the particular case 3.3). These mappings are expressed using concept bridges.

4.2 Phase 2: Evolving Ontology Properties Using Columns

After evolving ontology concepts using database tables, the next phase can start, where the ontology engineer identifies initial correspondences between ontology properties and the database columns. As in the previous phase, the initial correspondences are automatically treated to generate a set of ontology change operations and a set of mapping bridges.

There are different cases of initial correspondences between properties and columns. The distinction between these different cases is based on several criteria such as, the kind of the property (datatype or object property), the kind of the column (primary, foreign key or non-key column), the number (cardinality) of corresponding entities (properties/columns), and whether the correspondence is direct or using transformations.

We use the concept bridges established in the first phase to classify the initial correspondences. When a concept $C$ corresponds to a table $T$ (a mapping bridge exists between them), we can distinguish three categories of correspondence cases:

1. Correspondences between $C$ properties and $T$ columns.
2. Correspondences between $C$ properties and columns in other tables.
3. Correspondences between $T$ columns and properties of other concepts.
**Category 1**

In this category, we have a concept bridge between a concept C and a table T. We can distinguish the following seven cases of correspondences between C properties and T columns:

**Case 1:** A (datatype or object) property \( prop \) has no corresponding column (neither in T nor in any other table). In this case, nothing should happen, the ontology does not change, and nothing is added to the mappings document.

**Case 2:** A column \( col \) has no corresponding property in C. We distinguish two sub-cases. Firstly, if the column \( col \) is not a foreign key (case 2.1), then a datatype property \( dp \) corresponding to \( col \) should be added to the ontology. A property bridge between \( dp \) and \( col \) is thus added to the mappings (see case 2.1 of Figure 5).

Figure 5. Phase 2: Case 2: A column has no corresponding property

The second case (case 2.2) occurs when the column \( col \) is a foreign key \( fk \) referring to a column \( rc \) in another table \( RT \). We assume that the table \( RT \) is already mapped (in phase 1) to some (existing or created) concept \( D \) (there is a concept bridge between \( D \) and \( RT \)). The solution here is to add to the ontology an object property \( op \) whose domain is \( C \), and whose range is \( D \). This new object property corresponds to the foreign key \( fk \), therefore, a property bridge between \( op \) and \( fk \) is added to the mappings (see case 2.2 of Figure 5). This property bridge should:

- belong to the concept bridge between \( C \) and \( T \),
- refer to the related concept bridge which is between \( D \) and \( RT \),
- refer to the join expression that indicates how \( T \) and \( RT \) tables have to be joined together (\( T.fk = RT.rc \)).

**Case 3:** A datatype property \( dp \) corresponds directly to one column \( col \). In this case, the ontology does not change, and a property bridge between the property \( dp \) and the column \( col \) is added to the mappings.

**Case 4:** An object property \( op \) (from \( C \) to another concept \( D \)) corresponds directly to a foreign key \( fk \) referring to a column \( rc \) in another table \( RT \). We assume that the table \( RT \) is already mapped (in phase 1) to some (existing or created) concept \( F \). We distinguish two sub-cases. The first case (case 4.1) occurs if, fortunately, this concept \( F \) is the concept \( D \) (the range of \( op \)), then the object property \( op \) is mapped to the foreign key \( fk \), therefore, a property bridge between \( op \) and \( fk \) is added to the mappings (see case 4.1 of Figure 6). This property bridge should:

- belong to the concept bridge between \( C \) and \( T \),
- refer to the related concept bridge which is between \( D \) (which is \( F \)) and \( RT \),
- refer to the join expression that indicates how \( T \) and \( RT \) tables have to be joined together (\( T.fk = RT.rc \)).

Figure 6. Phase 2: Case 4: An object property corresponds directly to a foreign key

The second case (case 4.2) occurs if the concept \( F \) (mapped to \( RT \)) is not \( D \) (the range of \( op \)), then \( F \) is set to be a sub-concept of \( D \), and an object property \( op' \) is created from \( C \) to \( F \), and it is set as sub-property of \( op \).
The new object property $op'$ is mapped to the foreign key $fk$, therefore, a property bridge between $op'$ and $fk$ is added to the mappings (see case 4.2 of Figure 6). This property bridge should:

- belong to the concept bridge between $C$ and $T$,
- refer to the related concept bridge which is between $F$ and $RT$,
- refer to the join expression that indicates how $T$ and $RT$ tables have to be joined together ($T.fk = RT.rc$).

For example, let us consider a concept Conference that corresponds to a table conference. The table conference is related to another table city via the foreign key venue. The table city is mapped to a concept City. An object property hasLocation relates the concept Conference with another concept Location. The ontology engineer determined that the object property hasLocation corresponds to the foreign key venue. The solution is to set City as a sub-concept of Location, create an object property hasCity from Conference to City, set hasCity as sub-property of hasLocation, and make a property bridge between hasCity and venue.

**Case 5:** A datatype property $dp$ corresponds to a transformation Trans of one column $col$. In this case, the ontology does not change, and a single transformation property bridge between the property $dp$ and the transformation $Trans$ of the column $col$ is added to the mappings. For example, a datatype property fahrenheitTemp corresponds to a column celsiusTemp. The transformation could be an external function fahrenheit2Celsius that converts temperature from Fahrenheit to Celsius measures. The property bridge looks like: propertyBridge (fahrenheit2Celsius (fahrenheitTemp), celsiusTemp (celsiusTemp)).

**Case 6:** A datatype property $dp$ corresponds to a combination/transformation Trans of multiple columns $col_1, col_2, ... col_n$. For example, a datatype property name (single string) corresponds to the combination of two columns firstName and lastName. This combination is expressed using a transformation concatSpace that concatenates its first argument, a space and its second argument. For this case, there are two possible solutions:

1. We do not change the ontology, but we add a multiple transformation property bridge propertyBridge($dp$, Tran($col_1, col_2, ... col_n$)) to the mappings. For example: propertyBridge(name, concatSpace(firstName, lastName))

2. We split the datatype property into $n$ datatype properties $dp_i$, each of them corresponds directly to one column $col_i$. The Split_Property operation has to determine how each resulting datatype property is related to the old one. For example, when we split the name datatype property into firstName and lastName datatype properties, we may say:

   ```
   Split_Property(name AS
   firstName = name2FN(name),
   lastName = name2LN(name))
   ```

   where name2FN (respectively, name2LN) is a transformation giving the first (respectively, the last) name from a full name.

**Case 7:** A column $col$ corresponds to a combination/transformation of multiple datatype properties. In this case, the ontology is not changed, but a suitable single transformation property bridge is added to the mappings for each datatype property. For example, the column startingTime has the datatype datatime and it indicates the date and the time of the start of an event, so it corresponds to the combination of two datatype properties hasStartDate and hasStartTime. Consequently, the following property bridges is added to the mappings:

   ```
   propertyBridge(hasStartDate, datetime2date(startingTime))
   propertyBridge(hasStartTime, datetime2time(startingTime))
   ```

**Category 2**

As mentioned above, we identify correspondence cases according to the concept bridges established in the first phase to classify the initial correspondences. When we have a concept bridge between a concept $C$ and a table $T$, and when a $C$ property corresponds to a column in another table $U$, then we are in the second category. In this category, we can distinguish the following cases concerning datatype properties:

1. A datatype property $dp$ corresponds directly to one column $col$ in another table $U$.
2. A datatype property $dp$ corresponds to a transformation Trans on one column $col$ in another table $U$
3. A datatype property $dp$ corresponds to a combination/transformation Trans of multiple columns $col_1, col_2, ... col_n$ in the same table $U$ other than $T$.
4. A datatype property $dp$ corresponds to one column $colT$ in $T$ and to another column $colU$ in another table $U$.

Similar cases can also be distinguished for object properties.
Category 3

When we have a concept bridge between a concept C and a table T, and when a property of another concept D corresponds to a column of T, then we are in the third category. Some of this category cases are:
1. A column \( \text{col} \) corresponds directly to one datatype property \( \text{dp} \) of another concept \( D \).
2. A column \( \text{col} \) corresponds to a combination/transformation of multiple datatype properties.
3. A column \( \text{col} \) corresponds to an inherited datatype property.

However, because of the paper limits, the solutions of second and third categories are not presented.

5 Conclusion and Future works

We have proposed an ontology-evolution-based method to build ontologies from several heterogeneous information sources. In this method, an initial ontology is evolved progressively by involving an information source in each step. An evolution step includes a set of ontology change operations as well as a set of mapping bridges between the entities of the ontology being evolved and the entities of the source being involved. Two types of information source are considered, relational databases and XML documents. However, we have presented the case of relational databases only.

Involving a relational database consists of two main phases: evolving ontology concepts from database tables, and evolving ontology properties from database columns. In the second phase, several categories of correspondences are distinguished according to the established concept bridges. For instance, the currently used mappings include direct bridges, single transformation and multiple transformation bridges. We are currently working on identifying conditional correspondence cases. For example, we may say that a table \( \text{student} \) corresponds to a concept \( \text{Person} \) when the datatype property \( \text{job} \) has the value \( \text{student} \). This will add to the used specification the notion of conditional mapping bridges. This kind of bridges applies only when its associated condition holds.

An implementation of the method presented in this paper is currently under development.

References


A LITERATURE REVIEW ON EFFICIENT ONLINE ANALYTICAL PROCESSING SYSTEM

Sohail Asghar, Faraz Ahmad Osama

SZABIST Islamabad Campus, Department of Computer Science, Street No # 0 9, Plot No # 6, Sector # H - 8/4, Islamabad, Pakistan, sohail.asghar@szabist-ish.edu.pk, sanjolian@gmail.com

Abstract. OLAP (On Line Analytical Processing) is playing a vital role in today’s business world as well as in other domains. It gives users an analytical view of data resulting in effective decision making. Many domains are applying OLAP for different kinds of analysis due to its excellent performance and wide adaptability, such as analysis of network traffic and remote sensing images. With the advent of modern era, size of data is going to increase massively and hence there is a need of enhancement of OLAP efficiency. In the literature, different authors have acknowledged two major issues regarding efficiency of OLAP. These include cube computation and its aggregation, and management of data hierarchies. Keeping in mind all these aspects, we have summarized the literature and found the strengths and limitations in it. Critical evaluation of this literature reveals the research gaps for the researchers. We have combined effectively both summary and synthesis of literature.

Key words: OLAP, efficiency, cube computation, aggregation.

1 Introduction

Data is present in form of raw facts. These facts are needed to be developed in a view, after which it uncovers the hidden knowledge. OLAP provides the users an insight of data that is hidden openly in front of users and analysts [14]. OLAP gives view of data with different angles which users are unable to view with their natural abilities. It provides this manipulation of data where knowledge is easily and efficiently visible to the users.

Multidimensional data models are applicable effectively with the help of OLAP and its operation processing is applicable in data warehouses due to its requirement of special data organization, access method and implementation methods [14]. As data warehouses are maintained separately from operational databases so it fulfils the efficiency of OLAP

OLAP supports specific operations [15] such as rollup (to move up in the level of aggregation) drill down (to move down in the level of aggregation) slice and dice [14] for selection and projection and pivot (rotating the multidimensional view of data)

Data warehouses provides the facility to implement OLAP [14] on an extended relational DBMS which is called as ROLAP (Relational OLAP) servers as well as MOLAP servers (Multidimensional OLAP) in which multidimensional data is being directly stored on array data structures. HOLAP is also used which combines the features of both ROLAP and MOLAP and hence provides amazing efficient results.

All these facilities which OLAP is already providing, why do we need to make it efficient? Although the current OLAP is already performing well on the present data, there is a need to make OLAP more efficient as data is increasing more and more in size. New types of data also require OLAP to be more capable and efficient enough to fulfill the present day requirements.

How can we increase OLAP efficiency? The major problem which restricts the efficiency of OLAP is complex queries [6]. According to Wang et al. [3] OLAP queries can be executed efficiently by speeding up the queries with use of index structure or by operating on compressed data. Further to it, cube materialization and aggregation involves OLAP to decrease its efficiency and seems to be a bottleneck for OLAP.

Why are we studying these few techniques? The purpose of this research is to study deeply state of the art proposed techniques which are playing vital role in the research society. By summarizing the studied techniques along with critical evaluation of these, will open new appearance of research.

The rest of paper is organized as follows: Section II produces the literature review, section III provides critical evaluation of studied techniques, and section IV describes conclusion and future work.

2 Literature review

Literature review is an important part of research process. This section includes a discussion about the different techniques proposed in literature for evaluating OLAP efficiency.
Albrecht et al. [1] suggested the experimental OLAP server CUBESTAR which manages the multidimensional aggregates for efficient OLAP. Authors have proposed query processing based on algebra for multidimensional data. Relational database system has been used for this purpose. Strengths of their work are that they have pre-calculated queries for reuse in aggregate cache. Patch-working algorithm has been used to combine cached aggregates. Proposed extended star schema provides the facility of modeling of classification hierarchies. Limitations of their work are that for query caching a new query is required to be contained in old query which can increase processing time. Cost reduction and space requirements of almost 60% and less than 10% respectively have been achieved through prototypes.

Han et al. [2] proposed efficient computation of iceberg cubes with complex measures. Authors have used average measure as a base for their work and then implemented some other measures. Extensions of Apriori and BUC techniques have been proposed as Top-k Apriori and Top-k BUC. A new technique Hyper-Tree structure, called H-tree has been proposed. H-tree has been used in another new proposed technique Top-k H-Cubing for computing average iceberg cubes. The strengths of their work are recognition of some non-anti-monotonic conditions and their conversion into anti-monotonic conditions to prune search tree. Authors have confirmed previously out performed technique, Apriori, by BUC technique. Authors have proved through performance analysis that Top-k BUC and Top-k H-cubing are better than previously proposed techniques, even Top-k H-cubing outperforms Top-k BUC in some cases. The only limitation found in their work is that authors have not performed investigations on all the complex measures.

Wang et al. [3] suggested that efficient OLAP operations can be performed on spatial data using Predicate P-trees. The authors identified that spatial data is one of the most demanding for analysis purposes, which is required to be analyzed faster. Their major contribution is a new data warehouse structure PD-cube. It is a bit wise data cube in Peano order, which is consistent with Predicate P-Trees. The cube is partitioned into bit level to facilitate operations of predicate P-Trees. The strength of their work is that data converted in to bits allows fast execution of OLAP operations. It also supports compression of data and has performed efficiently on cube size > 1900 kb. The limitations of their work are that their experimentation is only on images taken from irrigation test area. Compression techniques has not been considered and studied. The architecture of their system has not been mentioned. In addition, there is no description of final view of data. The proposed method does not work efficiently on cube size < 1900 kb.

Akinde et al. [4] introduced efficient OLAP query processing in distributed data warehouse. This technique is anticipated for management of an IP network, by analysis on trace data, to verify network usage and behavior. Authors recognized the distributed nature of huge data collection which requires distributed data warehouse. The key strength of their work is distributed processing of complex and aggregate OLAP queries through GMDJ (Generalized Multi-Dimensional Join operator) expressions and skalla coordinator architecture. Authors introduced extended aggregation operator to express complex aggregation queries for this purpose. It is applicable on both star and snowflake schemas. The work done is limited to network trace data. It does not give any experimentation proof claimed for other domains maintained in data marts which are distributed within an enterprise. Furthermore, comparison has not been performed for conventional methods of trace data analysis.

Rao et al. [5] reported on the efficiency of OLAP query processing using spatial hierarchy in spatial data warehouses through pre-aggregation in cuboids. The term spatial index tree has been introduced for spatial indexing method. It is used by the authors as multidimensional set grouping hierarchy. Indexing technique GiST (Generalized Index Searching Technique) has been enhanced for OLAP favored search with application of heuristic estimation. The strengths of their work are the fast execution of query in spatial data by understanding its distinct capabilities by applying star schema in data warehouse. It reduces result set to 1/10th of the traditional query result. Besides, it also provides estimation of results correctness. The limitations of their work are that they have not tested the heuristic estimation on multiple types of data. It is applicable on only star schema. Bigger fan-out reduces the efficiency.

Feng et al. [6] claimed the efficient cube computation by exploiting data correlation Authors proposed a new technique range cubing. Range cubing generates a compressed data cube while keeping intact semantics and format of data. They presented correlation between dimensions as range trie, which is a tree structure. Range trie preserves dimensions and measures. Strengths of their work are preservation of rollup/drilldown semantics. Running time and storage space has been reduced simultaneously to a factor of 1/13 and 1/9 respectively when compared to H-Cubing, on real data sets. On synthetic data set, efficiency has also been improved. Limitations of their work are that comparisons have not been performed with other well established techniques like Apriori and BUC. Optimum level of compression has not been achieved.

Xu [7] introduced efficient OLAP query processing on Remote Sensing (RS) images. Author has proposed usage of OLAP to get insight of data so that data mining can be performed more efficiently. Data warehouse structure has been proposed for RS images which needs a metadata base and OLAP application server. ERDA Imagine V.8.5, MS Access and MS analysis services have been used for experimentation. Strengths of authors work are that simple tools have been used to propose a great scenario. Limitations of his work are that experimentation has not been performed on data obtained through OLAP for data mining.
Explanation is required that how data mining will be performed on pre-computed and materialized cubes. Efficiency of OLAP has been evaluated after pre-computation and materialization of cube. Data used is much small to be claimed for applicable in traditional data warehouse.

According to Morfonios et al. [8] OLAP efficiency can be improved using CURE (Cubing Using ROLAP Engine) technique. Authors have identified three problems which limit the OLAP efficiency and have proposed their solution. CURE proposes a traversing mechanism of an extended lattice to overcome increase in number of nodes in the cube. It presents an algorithm for partitioning of fact table to get it fitted in the memory. A new storage scheme to reduce redundant data has been proposed. Strengths of their work are that CURE is the first technique for construction of complete cube. APB-1 benchmark (12 GB) has been achieved. It applies compression on final results. Ability of iceberg cube construction is also present. It is applicable on large hierarchical data.

Timko et al. [9] proposed usage of probability distributions for enhancing efficiency of OLAP and hence Data warehouse (DWH). The authors have suggested the method of conversion of DWH into probability distribution. They have proposed a combination of interval and its associated probability for this purpose. A method of pre-aggregation has also been proposed for probability distributions. Higher level aggregated values have been achieved through it. Strengths of their work are that it provides a new way of getting estimated results using probability. The technique has shown efficiency on hyper-dynamic content. Limitations of their work are that by computation of expected values, important information can be lost. The technique is performed on only uniform data distribution. Coalescing used can reduce the precision of result.

Hu et al. [10] introduced a rapid dimension hierarchical aggregation algorithm on high dimensional OLAP. Authors have proposed to vertically partition a high dimensional data set into a set of disjoint low dimensional datasets which are called fragment mini cubes. Dimension hierarchical coding has been used for aggregation. Strengths of their work are that multi table joins have been reduced on large scale. The technique has outperformed other technique HCF (Hierarchically Clustered Fact Tables) both in computation time and storage space. Limitations of their work are that experimental comparison has been performed only with one other competitor technique. Description has not been provided about the framework on which requirements are performed. Computation of traditional data sets in traditional data warehouses has not been explained.

Hu et al. [11] reported a rapid group aggregation algorithm based on multidimensional hierarchical encoding. Authors proposed MDHEGA (Group Aggregation based on Multidimension Hierarchical Encoding) which is an improvisation of DHEPA (Pre-Aggregation based on the Dimension Hierarchical Encoding). Strengths of their work are that a new improved multidimension hierarchical coding technique has been proposed which further improve the efficiency of OLAP operations. Usage of buffer parameter in algorithm improves the aggregation of data. Limitations observed in their work are that experimental comparisons have not been given of previously introduced technique as well as other techniques introduced recently.

Chen et al. [12] proposed efficient OLAP using UDFs (User Defined Functions). UDF gives the functionality of writing user defined code and its usage as a standard SQL functions. Authors have proposed usage of UDF to generate OLAP cube within the UDF alongwith generation of association rule itemset. UDF can also be used to create user defined operators such as SQL cube operator. Strengths of their work are that cube generated using UDF remains in main memory until its results of aggregation are written in table, hence very efficient as compared to SQL. Experimental results have shown that UDF performs better both in efficiency for aggregation and association rules. UDF can be used with database without any integration problem.

<table>
<thead>
<tr>
<th>Author</th>
<th>Schema</th>
<th>Tree Structure</th>
<th>Cube formation</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Albrecht et al. [1]</td>
<td>Extended star</td>
<td>No</td>
<td>Yes</td>
<td>60 % increase</td>
</tr>
<tr>
<td>Han et al. [2]</td>
<td>Star</td>
<td>H-Tree</td>
<td>Iceberg</td>
<td>Much better</td>
</tr>
<tr>
<td>Wang et al. [3]</td>
<td>PD-Cube</td>
<td>Predicate P-Tree</td>
<td>Through PD- Cube</td>
<td>Good for cube size &gt; 1900 kb</td>
</tr>
<tr>
<td>Akinde et al. [4]</td>
<td>Star, Snow flake (Applicable on both)</td>
<td>No</td>
<td>Yes</td>
<td>Increased nearly half</td>
</tr>
<tr>
<td>Rao et al. [5]</td>
<td>Star</td>
<td>GiST</td>
<td>Yes</td>
<td>Better efficiency</td>
</tr>
<tr>
<td>Feng et al. [6]</td>
<td>Star</td>
<td>Range Trie</td>
<td>Range Cube</td>
<td>Reduced to less than 1/13th</td>
</tr>
<tr>
<td>Xu [7]</td>
<td>Star</td>
<td>No</td>
<td>Yes</td>
<td>10-50 % depending on memory size</td>
</tr>
<tr>
<td>Morfonios et al. [8]</td>
<td>Star</td>
<td>Bottom up</td>
<td>Through CURE</td>
<td>Much better in most of the cases</td>
</tr>
<tr>
<td>Timko et al. [9]</td>
<td>Not known</td>
<td>No</td>
<td>Yes</td>
<td>Efficient with approximate computation</td>
</tr>
</tbody>
</table>
3 Critical Evaluation

Authors have emphasized on construction of new tree structures. As shown in Fig 1, H-Tree proposed by Han et al [2] has special qualities such as scanning of database only once, tree and its header table contains complete information of iceberg cubes and it is completely compact. Unlike Han et al [2], Morfonios et al [8] has used bottom up approach for the tree formation and the original tree is divided into multiple small trees according to requirement. He has extended the tree structure of BUC [13]. BUC is one of the finest approaches in ROLAP which computes both complete and flat cubes. CURE technique [8] uses tree structure with slight change by capturing the relationship between different levels. It is accomplished by managing the recursive calls at the top of the tree. Feng et al [6] has proposed range trie, a new tree structure, which has been formed by evolving step by step. It is constructed from the set of data tuples in such a way that if input set is empty the tree will also be empty. If there is one tuple in the tree, the tree will be having one leaf which contains key as its all of the dimensions. Apart from all these tree structures Gist [5] supports the flexible index mechanism due to which it can provide extension to data as well as accessing of data. Hence, it has been used for spatial data. Another functional support it gives is implementation of one dimensional as well as multidimensional index. Spatial data tree structure is P-Tree (Peano tree) [3] is completely different from the above discussed tree structures as it is a quadrant based tree structure. It provides compression and very fast logical operations of bit sequential data.

Considering the aggregation methods proposed by different authors, Albreicht et al. [1] has used sum measure for the aggregation purpose whereas Han et al. [2] has used sum measure as well as count, avg, min and max. Now in the case of Akinde et al. [4] approach there is set of aggregate functions which are formed in GMDJ. Unlike above mentioned authors, Morfonios et al. [8] has distinguished the tuples as normal tuples, trivial tuples and common aggregate tuples for the purpose of aggregation. Timko et al. [9] has characterized the aggregation by implementing it on probability distribution. Hu et al. [10] and [11] has performed aggregation by doing pre-grouping transformation before using any joins in the data. Chen et al. [12] has performed cubing and aggregation in main memory and has compared it with traditional cube operator.

Star schema is the focus of most of the proposed techniques as shown in Table 1, whereas snowflake schema has been used by Akinde et al. [4], Hu et al. [10] and [11]. Akinde et al. [4] technique is applicable on star schema as well. DHEPA technique has been suggested firstly in [10] and then in [11] MDHEGA has been proposed which is based on firstly proposed technique. Both techniques have been implemented using hierarchical encoding.

Two of the techniques proposed by Timko et al. [9] and Chen et al. [12] have not used data warehouse schemas. First one has been implemented by converting the data values into probability and hence probability distribution and has been experimented in a java based prototype and second technique has been applied using a DBMS.

4 Conclusion and Future Work

OLAP enables knowledge workers and managers to analyze data and take necessary decision on its basis. As data and data types are increasing day by day, need for efficiency enhancement is also increasing.

A thorough and deep study shows some new scenarios for the enhancement of OLAP efficiency. Most of the research work has been focused on Tree formation and its traversal for fast execution of OLAP queries. The objective of this paper is to provide the critical evaluation of most prominent technical papers available for efficient OLAP.

Techniques given by Albrecht [1], Han [2] and Xu [7] have been proved efficient according to the results given in table 1. We are intending to develop a new technique for efficient OLAP in future on the basis of limitations found in Xu’s [7] work. Data mining is a strong field which can lead to better results in OLAP, as claimed in [7]. Our focus is to use data mining for developing an efficient OLAP system.

References


Figure 1: Evaluation techniques for efficient OLAP

- 390 -