Supporting Ontology-based Dynamic Property and Classification in WebSphere Metadata Server

Shengping Liu[1], Yang Yang[1], Guotong Xie[1], Chen Wang[1], Feng Cao[1], Cassio Dos Santos[2], Bob Schloss[3], Yue Pan[1], Kevin Shank[2], John Colgrave[2]
[1]IBM China Research Laboratory
Building 19A, Zhongguancun Software Park, Beijing, 100094, China
{liusp, yangyy, xieguot, chwang, caofeng}@cn.ibm.com
[2] IBM Software Group
50 Washington Street, Westborough, MA 01581, USA
{scdos,kshank}@us.ibm.com，colgrave@uk.ibm.com
[3] IBM Watson Research Center
P.O.Box 704, Yorktown Heights, NY 10598, USA
rschloss@us.ibm.com

Abstract. Metadata management is an important aspect of today's enterprise information systems. Metadata management systems are growing from tool-specific repositories to enterprise-wide metadata repositories. In this context, one challenge is the management of the evolving metadata whose schema or meta-model itself may evolve, e.g., dynamically-added properties, which are often hard to predict upfront at the initial meta-model design time; another challenge is to organize the metadata by semantically-rich classification schemes. In this paper, we present a practical system which provides support for users to dynamically manage semantically-rich properties and classifications in the IBM WebSphere Metadata Server (MDS) by integrating an OWL ontology repository. To enable the smooth acceptance of Semantic Web technologies for developers of commercial software which must run 24 hours/day, 7 days/week, the system is designed to consist of integrated modeling paradigms, with an integrated query language and runtime repository. Specifically, we propose the modeling of dynamic properties on structured metadata as OWL properties and the modeling of classification schemes as OWL ontologies for metadata classification. We present a natural extension to OQL (Object Query Language)-like query language to embrace dynamic properties and metadata classification. We also observe that hybrid storage, i.e., horizontal tables for structured metadata and vertical triple tables for dynamic properties and classification, is suitable for the storage and query processing of co-existing structured metadata and semantic metadata. We believe that our study and experience are not specific to MDS, but are valuable for the community trying to apply Semantic Web technologies to the structured data management area.

1 Introduction

Metadata is pervasive in large enterprises and can be thought as the “DNA” of enterprise applications. The structured metadata is not only descriptive information about data, but prescriptive information that constrains the structure and content of data. The metadata can be technical metadata, such as relational schemas, XML schemas, schema mappings, UML models and application interface specifications, and can be business metadata, such as business concepts, business rules and business process definitions in an enterprise. The metadata management tool (also known as repository) [15] is crucial for enterprise information management and has become the foundation of Data Warehousing [9], Enterprise Information Integration [8] and Service-Oriented Architecture (SOA).

Recent standards work on MOF/XMI [11] within OMG for metadata representation and interchange has been followed by many vendors, then MOF-based metadata repositories have become the mainstream in industry offerings 5]. Amongst these MOF-based metadata repositories, a common feature is the object-oriented storage strategy where Object-Relational Mapping functionality is used to generate physical schemas for the corresponding MOF meta-models and provide an object-oriented programming interface to the underlying database. One typical example is the IBM WebSphere MetaData Server (MDS), which is a unified metadata services infrastructure within a service-oriented architecture.

Within the enterprise-wide IT environment, metadata management has become more and more challenging because of rapidly-changing business requirements. Metadata repositories are growing from tool-specific, application-specific systems to enterprise-wide, asset-management and architecture decision support systems, in which metadata are shared and integrated across multiple applications or even third party tools [6]. While the metadata and their relationships dramatically grow, it is impossible to design a unified meta-model for all kinds of metadata with all possible attributes and relationships at design stage as the business requirements evolve. Therefore there is a requirement to dynamically add properties for classes in the registered metamodel. For example, after a WSDL meta-model which describes WSDL documents has been registered, a service administrator might add QoS (Quality of Service) metadata to the "WSDLService", such as the "responseTime". Another example is to dynamically build particular relationships across registered meta-models. After the metadata repository has run and collected entries for a period of time, a user needs to create a dynamic relationship "dependsOn" from the class "Activity" in the business process meta-model to the class "WSDLService" in the WSDL meta-model, which later can be used to enable traceability and impact analysis across those models. Moreover, semantic annotations are required to enrich the semantics of dynamic relationships, e.g. annotating "dependsOn" as "transitive". Based on these semantic annotations, ontology reasoning will be made to infer additional information which is not explicitly defined.

In metadata management, a classification scheme is used to classify the metadata objects, such as relational schemas and WSDL definitions in a metadata repository. Examples of classification schemes range from simple tags (keywords), thesauri, taxonomies to formal ontologies. With the growing volumes of metadata in different applications and users of metadata from various business units of enterprises, a flexible and semantic-rich classification scheme is needed to help different users to organize metadata from different viewpoints. This is because: (1)the classification scheme itself needs reasoning on the classifier hierarchy; (2)users need to define rich expressions on the classification scheme to declare dynamic classifiers, in addition, the expression can be defined on dynamic properties. For example, after the dynamic property "dependsOn" is declared, user can define "DataDependentService" as a new classifier, which contains the WSDL services that depend on a "DataService".

With the emergence of the Semantic Web [14], Web metadata markup languages, i.e. RDF (Resource Description Framework) [1] and OWL (Web Ontology Language) [3], have become W3C Recommendations. RDF originated from the W3C Metadata Activity, and is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a Web page [2]. The most important feature of the RDF data model is that it treats properties as first-class citizens and allows them to be attached to a class dynamically. OWL is a formal logic language to define the vocabularies in RDF documents. It is intended to represent structured metadata ranging from a simple taxonomy, a thesaurus, or to a formal ontology. In practice, OWL is an emerging standard to represent the classification schemes, because of its rich expressivity, formal semantics and reasoning capabilities. Therefore, it is natural to apply the Semantic Web technologies, namely RDF and OWL, to meet the emerging requirements of enterprise-wide metadata management.

In this paper, we propose a practical system which supports (OWL) ontology-based dynamic properties (i.e. dynamic attributes and dynamic relationships) and metadata classifications in the IBM WebSphere Metadata Server (MDS) by integrating an ontology repository. To enable smooth acceptance of Semantic Web technologies for developers of commercial software that must run continuously, the system is designed to consist of integrated modeling paradigm, query language and runtime repository. Our contributions of this work can be summarized as below.

(1) We use the semantic web languages, RDF and OWL in particular, to extend the MOF/XMI based metadata language with more flexibility and semantics. Dynamic attributes and relationships are represented as datatype properties and object properties in OWL respectively, and classification schemes are represented as OWL ontologies.

(2) We extend the object-oriented query language of MDS, XMQL, with additional classification query functions and dynamic property extensions. Users can issue hybrid queries over structured metadata, dynamic properties and semantic classifications simultaneously. The hybrid query follows the language convention of XMQL. MDS users do not need to learn another query language for RDF/OWL, i.e. SPARQL [12]. Ontology reasoning will automatically be conducted when answering queries.

(3) We develop a hybrid storage system by integrating a state-of-the-art ontology repository, namely the SOR Repository [16] to MDS, which is deployed in the same database with the repository of MDS. Specifically, information of dynamic properties and classification are stored in the SOR repository. Hybrid queries will be split to MDS queries and ontology queries, with the SQL fragments translated by MDS and SOR, and subsequently merged and executed by the relational database engine.

The rest of this paper is organized as follows. Section 2 describes the architecture of the integrated system. Section 3 presents the modeling and usage for ontology-based dynamic properties and classifications. Section 4 introduces the metadata query language XMQL with extensions and the query processing engine. Section 5 presents the use study on terminology services implementation. Section 6 discusses related work. Finally Section 7 summarizes the contributions of this paper.

2 System Architecture

To support ontology-based dynamic property and classification in metadata management, one approach is to store all the structured metadata and data on dynamic properties and classifications into an ontology repository. However, most of the current RDF databases (triple-stores) scale poorly since most queries require multiple self-joins on the vertical triples table [21], and the large volumes of structured metadata in enterprise are commonly stored in horizontal tables in a relational database. So we chose the hybrid approach that integrates the MDS with an ontology repository SOR [16], i.e., the structured metadata are still stored in the horizontal tables of MDS and the data on dynamic properties and classifications are stored in the vertical triples table of SOR. This system is called MDS++. Before presenting the detailed system architecture, we give a short introduction to both MDS and SOR.

2.1 WebSphere Metadata Server

The IBM WebSphere Metadata Server (MDS) is a unified metadata services infrastructure that's designed to ease metadata management, access, and sharing within a service-oriented architecture. MDS is available as part of the IBM Information Server platform. It provides metadata management services to products in the IIS platform and is additionally used as a common metadata services infrastructure for metadata products in other IBM software brands.

MDS was built with the Eclipse Modeling Framework[1] (EMF), which is a modeling framework and code generation facility for building tools and other applications based on a structured data model. In general, the design of this product is following the Object-Relational Mapping (ORM) paradigm to manage metadata objects, similar to the well-known ORM tool Hibernate[2]. When a meta-model is registered with the MDS at MDS build time, the CRUD (Create, Read, Update and Delete) API and the relational persistence schema will be automatically generated by EMF tools. Then MDS can support query and persistence of metadata that are instances of this meta-model.

2.2 SOR Repository

SOR (Semantic Object Repository) is a high performance OWL ontology repository built on relational databases. SOR translates OWL semantics into a set of rules which can be easily implemented using SQL statements on RDBMS. SOR supports the RDF data query language SPARQL. In SOR, a SPARQL query is first translated into a single SQL statement which is evaluated on both explicit assertions and inferred results materialized in the persistent store, benefiting from decades of relational database optimization. The following two features of SOR are critical for the integrated system.

[1] Eclipse Modeling Framework：http://www.eclipse.org/emf/

[2] Hibernate: Relational Persistence for Java and .NET. http://www.hibernate.org/

(1) The inferred facts are materialized when loading the data, which can improve the response time of queries.

(2) The SPARQL query is translated into a single SQL statement, which can be executed over the repository directly or embedded as a sub-query of other SQL queries.

2.3 System Architecture of MDS++

To enable smooth acceptance of Semantic Web technologies for developers of commercial software that must run continuously, the system is designed to consist of integrated modeling paradigm, query language and runtime repository. Fig. 1 shows the overall system architecture.

For the modeling paradigm, MDS was developed using Model-Driven Architecture and is based on EMF. The SOR architecture follows the model-driven approach for ontology engineering [18]. In this approach, the RDF and OWL is defined based on Ontology Definition Metamodel (ODM)[3]. The ODM specification is implemented by EODM[4], based on the EMF framework with additional inference and model transformation functions. From Fig. 1, we can see that MDS and SOR provide a unified EMF view for users to access the metadata through query and CRUD API.

For the query language, we extend the XMQL query language of MDS to XMQL++, with additional classification query functions and dynamic property extensions. Users can simultaneously query over the structured metadata stored in MDS, the information about dynamic properties and classifications stored in SOR.

For the runtime repository, we make the ontology repository tightly-coupled with the MDS repository, i.e., the tables of ontology repository will be deployed in the same database with MDS and are visible to MDS. The two repositories are connected by the unified object identifier in MDS, which is also used as the internal identifier for the RDF resource in SOR. When a hybrid query is issued, the system will translate the query to one single SQL query which will access the tables of both repositories simultaneously. The advantage of this approach is that it provides high performance because queries are translated to SQL queries which can fully utilize the optimization provided by the underlying DBMS.

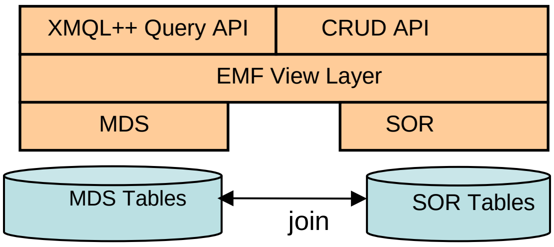


Fig. 1. The system architecture of MDS++

[3] Ontology Definition Metamodel Specification, www.omg.org/docs/ad/05-08-01.pdf

[4] EODM homepage on Eclipse, http://www.eclipse.org/modeling/mdt/?project=eodm

3 Dynamic Property and Classification Scheme

In this section, we will show how to model and use the ontology-based dynamic properties and ontology-based classification in MDS++.

3.1 Ontology-based Dynamic Property

Compared to static EMF properties, the term “dynamic” implies that this kind of property can be declared and attached to the meta-model after the meta-model is registered with the repository. Dynamic properties can be divided into dynamic attributes and dynamic relationships. A dynamic attribute describes some kind of attribute of model elements. The domain of a dynamic attribute must be an EClass. The range of a dynamic attribute can be the supported data types in EMF, such as EString, EInt, etc. A dynamic relationship describes some kind of relationship between model elements. The domain and range of a dynamic relationship must be an EClass. The domain and range constraint on the dynamic properties is an important design consideration to guarantee that the dynamic properties are operated similarly to the static EMF properties.

Because the dynamic properties are treated as properties in OWL ontology, semantic annotations can be further added to enrich their semantics. Currently four kinds of semantic annotations borrowed from the OWL language can be supported: symmetric, transitive, inverseOf and subPropertyOf. After the dynamic properties are declared, the user can fill in the values for these properties. As an example, after the meta-model for WSDL documents, as shown in Fig. 2, is registered with the MDS, a service administrator can create a new dynamic attribute “businessFunction” and declare its domain as the class “WSDLService” in the meta-model and its range as Ecore data type “EString”. Similarly, the service administrator can create the dynamic attribute “responseTime” and “serviceStatus”, and create the dynamic relationship “dependsOn” to describe the dependency relationships among services, even if this relationship is not modeled in the original meta-model. In addition, the user can declare that the “dependsOn” relationship is transitive. The client tool UI to build the dynamic properties is shown in Fig. 3.

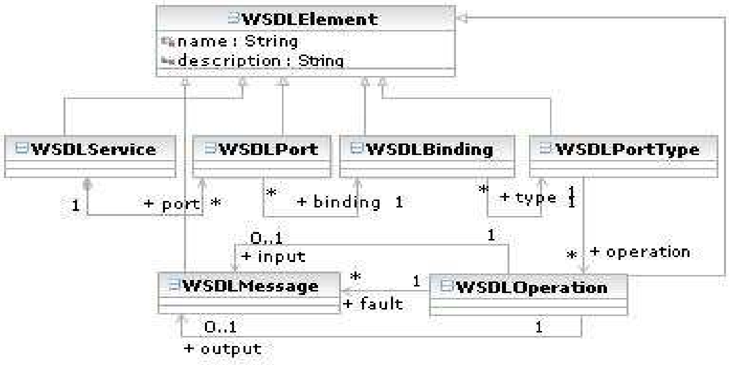
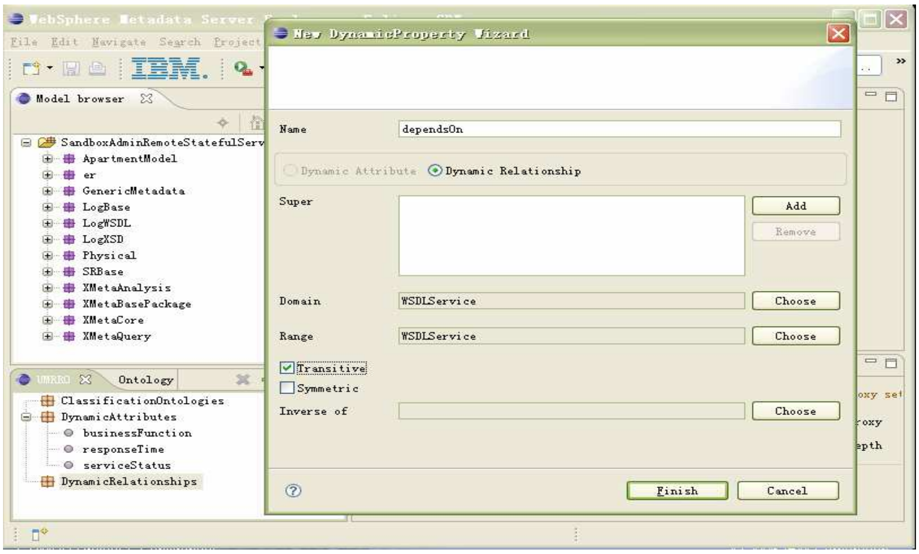


Fig. 2. A simplified meta-model for WSDL Document



Name: dependsOn

○ Dynamic Attribute ● Dynamic Relationship

Super:

Domain: WSDLService

Range: WSDLService

☑ Transitive

□ Symmetric

Inverse of:

Fig. 3. An eclipse-based UI to declare dynamic properties

3.2 Ontology-based Classification

In MDS++, a classification scheme is represented by an OWL ontology and the task of classification is supported by the built-in OWL reasoner of SOR. The classification scheme can be created in two ways. One is to load an existing OWL ontology as a classification scheme. In this way, only the named OWL classes inside the ontology will be taken as classifiers. Those anonymous OWL classes or expressions can not be classifiers, because they do not have URIs to get them identified. Another way is to build a classification scheme from scratch by using APIs. A user can create a classifier as a named class in the OWL ontology and setup the explicit hierarchy using the OWL construct “subClassOf”. In addition, users can define the new classifier using OWL constructs supported in SOR: intersectionOf, and OWL restrictions on dynamic properties: someValuesFrom and hasValue.

After the classification scheme is built, a user can manually classify some metadata to classifiers as shown in Fig. 4. Then OWL reasoning can be applied to find the implicit classification information, eg. Find all metadata classified by a high-level class. In addition, based on the semantics of the OWL class expressions and restrictions, automatically classification can be made according to values of dynamic properties by the OWL reasoner. For example, the classifier “DataDependentService” can be defined as: *someValuesFrom(dependsOn, DataService)*. If a service s1 dependsOn s2 and s2 is a DataService, then s1 can be automatically classified as “DataDependentService” without explicit declaration.

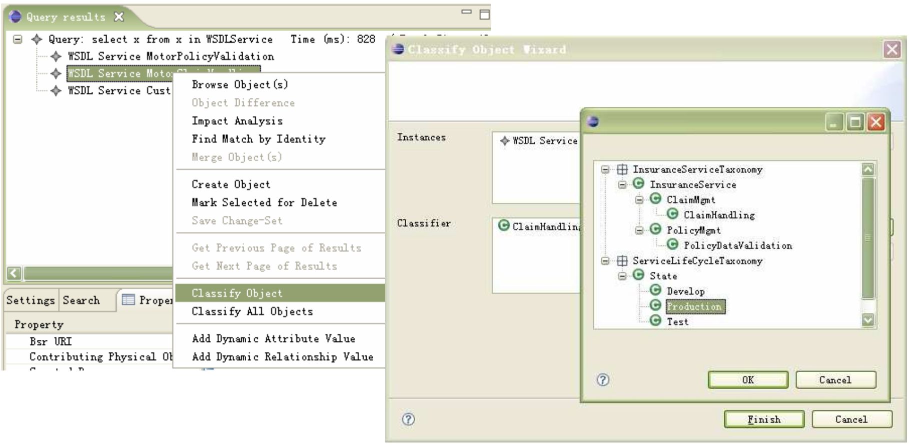


Fig. 4. Classify metadata object to classifier in classification scheme

4 XMQL++ Query Language and Processing

Queries written in the query language XMQL are the main access points to the metadata stored in MDS. Thus, how to let MDS users access the dynamic properties and classification information though query was a central design problem for MDS++. One alternative we explored was to design a hybrid query language that embeds SPARQL query into the XMQL query language by a predefined function, similar to the *RDF_MATCH* function introduced by Oracle 10g [17]. This design choice was not accepted by the MDS product engineers because it is too complex for MDS users and tool developers to learn two kinds of query languages. Therefore, we designed the query language XMQL++ with a minor extension of XMQL to enable hybrid queries over static metadata, dynamic properties and semantic classifications simultaneously. We will illustrate the extension with simple query examples and introduce the query processing mechanism in this section. The formal specification of XMQL++ and technical details of query processing are omitted due to limited space.

4.1 A Short Introduction to XMQL

XMQL, a subset of ODMG's OQL[5], is the query language of WebSphere Metadata Server (MDS). It is a general purpose SQL-like declarative query language with special features designed for the efficient retrieval of instances stored in an MDS repository. A basic structure of an XMQL query is the *select-from-where* clause. The *select* clause defines the structure of the query result, which can be either a collection of objects, or a collection of rows with the selected attribute values. The *from* clause introduces the primary class extent(s) against which the query runs. A class extent is

[5] http://www.odmg.org/

the set of all instances of a given class. A variable needs to be declared to iterate over the instances of each class being queried. The *where* clause introduces a predicate that filters the instances of the collections being queried. The XMQL query adopts path expression to denote the traversal of a reference from one object to another, using the "->" operator or the access of an attribute using the "." operator. For example,

```
SELECT e.name
FROM  e IN Employee, p IN e->workForProject
WHERE e->workAt.country="US" AND
   p.name="NewHotel"
```

This query returns a set of rows containing each an employee name for the employees located in the United States that work for the project named "NewHotel".

4.2 Extensions for Dynamic Properties

Dynamic properties play the same role as ordinary EMF properties from user's view, though they are stored independently in separate repositories. To be compatible with the design principal of XMQL, the domain and range of any dynamic property must be explicitly declared in the design stage. Then XMQL query compiler can get the type information of dynamic properties when processing queries. For example, "y in x→*dependsOn*" will be of type "WSDLService".

The following example queries show how dynamic properties are used in XMQL++. The query to get all WSDL services's response time which are dependent on "service1" can be written as (the dynamic properties are in Italic fonts):

```
SELECT  x.name, x.responseTime FROM x IN WSDLService,
        y IN x→dependsOn WHERE y.name ="service1"
```

4.3 Extensions for Classification

Classification functions are provided to enable queries over classification information in XMQL. The basic classification functions are listed as below.

- Object[] *classifiedBy*(URI): It will return a list of objects that are classified to the class represented by the URI argument. This function can be used as an argument of an "IN" predicate in a "WHERE" clause of XMQL.
- URI[] *classifiers*(pathExpression): It will return a list of URIs which are classifiers of the objects denoted by the path expression.

The following example queries show how classification functions are used in XMQL++. The query to get all WSDL services which are classified to "ClaimMgmt" can be written as:

```
SELECT  x FROM x IN WSDLService,
   WHERE x IN classifiedBy("http://foo.org/#ClaimMgmt")
```

The query to get all classifiers for WSDLService "service1" can be written as:

```
SELECT classifiers(x) FROM x IN WSDLService
   WHERE x.name ="Service1"
```

4.4 XMQL++ Query Processing

In MDS, an XMQL query is translated to a single SQL query at compile time, and the results are returned after executing the SQL query by the underlying relational database engine. Similarly, a XMQL++ query which needs to access objects and ontology related information simultaneously will also be translated to one single SQL. All invocations of dynamic properties and classification functions in XMQL++ will be translated to SPARQL queries during the query translation. Fig. 5 shows the high level workflow of XMQL++ query processing.

To translate a XMQL++ query into SQL query, the XMQL translator firstly will find out all the ontology related invocations. It then will pass these invocations to the *TripleQueryHandler*. The *TripleQueryHandler* will translate the invocations into SPARQL queries and submit them to *SOR SPARQL Engine*. The *SOR SPARQL Engine* will answer those queries by returning SQL sub queries whose results are the answers of the ontology invocations. Finally, the *XMQLTranslator* will merge these sub queries with the SQL query from O-R Mapping.

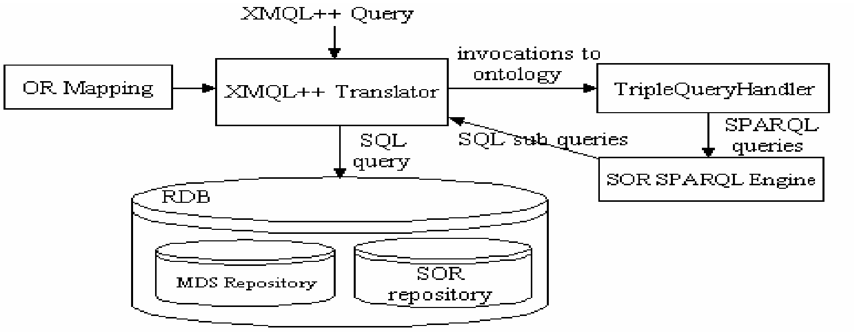


Fig. 5. XMQL++ query translation process

5 Use Study on China Healthcare Solution

Code system (terminology) is an important kind of metadata in healthcare applications, because consistent medical terminology is essential for the sharing, exchange and integration of healthcare information [20]. MDS++ is currently used in the IBM China Healthcare Solution to provide terminology services for healthcare industry. The terminology service is an implementation of HL7 Common Terminology Services (HL7 CTS)[6], which is an Application Programming Interface (API) specification that is intended to describe the basic functionality that will be needed by HL7 Version 3 software implementations to query and access terminological content. CTS API includes two parts of API: a message API that is specific to HL7 software, and vocabulary API, which is general to allow applications

[6] HL7 CTS Specification, http://informatics.mayo.edu/LexGrid/index.php?page=ctsspec

to query different terminologies in a consistent, well-defined fashion. The current implementation supports the vocabulary API based on the model[7] shown in Fig. 6.

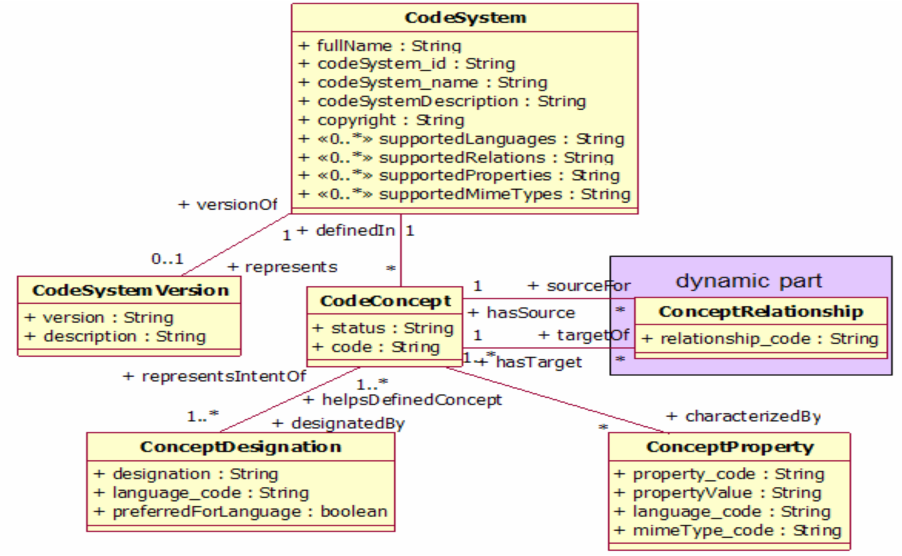


Fig. 6. The reference vocabulary model for CTS implementation

In this model, the class *ConceptRelationship* class represents binary relationships over the set of *CodedConcept*s defined in a single *CodeSystem*. Two of the supported relationships in CTS are “isSubtypeOf” and “isPartOf”, which are transitive and need inference support for computing the transitive closure. To register this model to MDS++, we divide the model as two parts: one is static part except the class *ConceptRelationship*; another is the dynamic part that models every “relationship_code” as a dynamic relationship with both domain and range as *CodeConcept*. The instances of the static part will be stored as horizon tables in WMS and the instances of the dynamic part will be stored as vertical triple tables in SOR repository to leverage SOR’s reasoning capability.

We have loaded four kinds of healthcare code systems into MDS++: LOINC, ICD-10, TCM(Terminology of Traditional Chinese Medicine) and SNOMED CT [19]. When loading SNOMED CT, we find that the model of SNOMED CT has additional information not captured by the static part of this vocabulary model: the *ConceptDesignation* has three categories: “fully-specified name”, “preferred name” and “synonym”. To fully keep the information in SNOMED CT, we add one dynamic attribute *designationCategory* with the domain as the class *ConceptDesignation* and the range as an enumeration of the three allowable string values.

We have implemented the CTS Vocabulary APIs by transforming the API calls to XMQL++ queries. For example, for the runtime API call to determine whether two concept codes (e.g. 25064002 and 279001004) in SNOMED CT are related via the relationship “isSubtypeOf”, the corresponding XMQL++ query is:

[7] This model does not support “relationship qualifiers” as appeared in the CTS API, because our supported code systems do not have “relationship qualifier” for any relationship.

```
SELECT COUNT(x) FROM x IN CodeConcept,z IN x->definedIn,
  y IN x->isSubtypeOf  WHERE  ( x.code=”25064002” AND
  y.code=”279001004” AND z.codeSystem_name=”SNOMED CT” )
```

Based on the experience on implementation of CTS on MDS++, we observe that there are three features essential for the acceptance of the integration of an ontology repository to an industrial-strength metadata repository:

(1) Model flexibility: users/programmers can attach any attributes or relationships to classes in the registered model without redeployment of the model. For example, the *designationCategory* attribute can be added for the class *ConceptDesignation.*

(2) Reasoning support: programmers need not write additional code to handle the complex reasoning problems, such as the transitive closure computing.

(3) User-friendliness: the MDS users write similar APIs and use our modestly-extended query language to handle on dynamic properties and classifications. The underlying Semantic Web technologies are mostly transparent for the MDS users/programmers.

Another encouraging observation is that the hybrid storage pattern, i.e., horizontal tables for the part of structured metadata with no reasoning involved and vertical triple tables for dynamic properties or the part of metadata with reasoning support, is effective and efficient for the storage and query processing of complex metadata that needs reasoning support on part of the metamodel. In a broader sense, the hybrid storage pattern is promising to manage the co-existing structured metadata and semantic metadata.

However, we also observe that this approach can not leverage the full power of Semantic Web technologies, for example, the expressive SPARQL query constructs, such as DESCRIBE query, which is really applicable for the implementation of the CTS API to return a complete description of a coded concept, and also the named graph support, which is useable for modeling a code system as a named graph.

6 Related Work

Metadata management has a long history within the evolving discipline of data management, and the recent focus is on the integration of diverse metadata and MOF-based metadata repositories [5]. Some notable examples are MetaMatrix [10], NetBeans Metadata Repository[8]. As far as we know, both of these metadata repositories do not support the adding of semantic properties to classes in the meta-model at runtime. There is a simple model extension approach that every class has a list of <property, value> pairs each with a name and appropriate value. Another existing approach to add links or relationships between different models is using weaving models [7]. A weaving model conforms to a weaving meta-model, in which a link have multiple link ends that each holding a reference to a model element. For different application scenario, e.g, traceability and schema mapping, the meta-model must be extended and the links can have different semantics. Compared to both

[8] NetBeans Metadata Repository: http://mdr.netbeans.org/

approaches, our proposed approach is not just about modeling dynamic properties, but is a system that has unified representation, storage, query and reasoning for dynamic properties. From the modeling perspecitve, OWL is much more expressive and has formal semantics for representation of various kinds of attributes and relationships. From the usage perspective, our approach hides the complexity of management of the <property value> pairs or the weaving links. For the end-user, dynamic properties are accessed similar to the ordinary properites in object-oriented model.

In our approach, a traditional relational database is integrated with a RDF-based triple store to support metadata storage. This kind of hybrid storage is also supported in Oracle 10g [17]. It introduces a SQL table function *RDF_MATCH* that embeds a SPARQL query to query RDF data. Then users can write SQL queries with joins between variables in table function and columns in the relational data. In this implementation, end users have to understand all the complexity of SPARQL query language and write complicated joints by themselves. In contrast, our approach hides the complexity to end users by slightly extending the XMQL language. MDS++ engine will handle joints between static repository and ontology repository because they use the unified object identifier as the shared identifier. The MDS users and MDS tool programmers did not need to learn any query languages specific to RDF.

7 Conclusion

Metadata management systems are growing from tool-specific repositories to enterprise-wide metadata repositories, at the same time, more and more semantic-rich metadata like RDF and OWL ontologies are emerging. The structured metadata and semantic metadata would co-exist in enterprises, and their “marriage” would address some of the key challenges of enterprise-wide metadata management. Traditionally, these two communities, MOF/XMI based structured metadata and RDF/OWL based semantic metadata, have developed their own standards and tools that are incompatible. However, enterprises require a more comprehensive metadata management environment. We believe our work demonstrates a practical architecture moving repositories towards that direction. In our work, the practical system MDS++ provides support for users to manage semantic-rich properties and classifiers in the IBM WebSphere Metadata Server (MDS) by integrating the ontology repository SOR. Our experience can be summarized as this: to enable the smooth acceptance of Semantic Web technologies for commercial metadata product developers with existing products, the integrated system must consist of integrated modeling paradigm, query language and runtime repository. This is essential for the successful application of MDS++ for China Healthcare Solution.

References

1. Manola F. and Miller E.. RDF primer. W3C recommendation, Feb 2004.
2. Brickley, D. and Guha, R.V. RDF vocabulary description language 1.0: RDF schema. W3C recommendation, Feb 2004.

3. Smith M. K.,Welty C., and McGuinness D. L.. *OWL web ontology language guide*. W3C recommendation, Feb 2004

4. Baker, N.L., Le Goff, J-M: *Meta Object Facilities and their Role in Distributed Information Management Systems*, In Proc. of the EPS ICALEPCS97, 1997.

5. Arun Sen, Metadata management: past, present and future, *Decision Support Systems*, 37(1), 2004.

6. Ganesan Shankaranarayanan and Adir Even, *The Metadata Enigma*, CACM, 49(2), 2006.

7. V. Stefanov, B. List. Business Metadata for the Data Warehouse: Weaving Enterprise Goals and Multidimensional Models, *Models for Enterprise Computing 2006 - International Workshop at EDOC 2006*.

8. T. Thangarathinam, G. Wyant, J. Gibson, and J. Simpson. Metadata management: the foundation for enterprise information integration. *Intel Technology Journal*, 8(4), 2004.

9. Robert Muller, Thomas Stohr, and Erhard Rahm. An integrative and uniform model for metadata management in data warehousing environments. *In International Workshop on Design and Management of Data Warehouses*, pages 12–28, 1999.

10. R. Hauch, A. Miller, and R. Cardwell. Information intelligence: metadata for information discovery, access, and integration. In *the 2005 ACM SIGMOD*, pages 793 – 798, 2005.

11. Object Management Group: Meta Object Facility Specification, version 1.4, OMG document formal/02-04-03

12. E. Prud'hommeaux, A. Seaborne. SPARQL Query Language for RDF, W3C Recommendation 15, January 2008.

13. J.A. Zachman. "A Framework for Information Systems Architecture," *IBM Systems Journal*, 26(3), 1987.

14. T. Berners-Lee, J. Handler, and O. Lassila. The semantic web. *Scientific American*, 184(5): 34–43, 2001.

15. Marco, D., Building and Managing the Meta Data *Repository: A Full Lifecycle Guide*, John Wiley & Sons, Inc., New York, 2000.

16. L. Ma, L. Jing, C. Wang, F. Cao and Y. Pan. *Effective and Efficient Semantic Web Data Management over DB2*, in Proc. of the 28th ACM SIGMOD Conference, 2008.

17. Chong E. I., Das S., Eadon G., Srinivasan J.: *An Efficient SQL-based RDF Querying Scheme*, in Proc. of VLDB 2005, pp.1216-1227, 2005.

18. Yue Pan, Guotong Xie, Li Ma, Yang Yang, ZhaoMing Qiu, and Juhnyoung Lee. Model-driven ontology engineering. *Journal of Data Semantics* VII, pages 57–78, 2006.

19. SNOMED Clinical Terms. Northfield, IL: College of American Pathologists, 2007.

20. Rector AL, Solomon WD, Nowlan WA, *et al.* A Terminology Server for Medical Language and Medical Information Systems. *Meth Inform Med* 1995, 34(1-2):147-57.

21. Abadi, D., A. Marcus, et al.. *Scalable Semantic Web Data Management Using Vertical Partitioning*. In Proc. of VLDB 2007, pp. 411-422, 2007.