EXTENSIBLE METADATA REPOSITORY FOR INFORMATION SYSTEMS AND ENTERPRISE APPLICATIONS

Ricardo Ferreira
UNINOVA – Instituto de Desenvolvimento de Novas Tecnologias, Quinta da Torre, 2829-516 Caparica, Portugal

João Moura-Pires
CENTRIA/FCT, Quinta da Torre, 2829-516 Caparica, Portugal

Keywords: Metadata, Repository, XML, Information Systems, Enterprise Applications.

Abstract: Today's Information Systems and Enterprise Applications require extensive use of Metadata information. In Information Systems, metadata helps in integration and modelling their various components and computational processes, while in Enterprises metadata can describe business and management models, human or physical resources, among others. This paper presents a light and no-cost extensible Metadata Repository solution for such cases, relying on XML and related technologies to store, validate, query and transform metadata information, ensuring common operational concerns such as availability and security yet providing easy integration. The feasibility and applicability of the solution is proved by a case study where an implementation is running in operational state.

1 INTRODUCTION

In the context of this article, metadata (data about data) is understood as any information needed to develop and maintain an Information System (Vaduva and Dittrich, 2001). Metadata Information can be classified in several types and groups (Tannenbaum, 2002): business metadata (for modelling business or domain subjects) and operational metadata (for technical details on operations and processes), which can be further categorized in *documentation metadata* and *operational metadata*. With the increasing complexity of Information Systems in the last years, metadata started to gain its importance and acceptance, since it provides a backbone for systems integration, management, evolution and documentation.

With its raising importance, metadata standards appeared as a solution for solving common problems enabling sharing and interoperability of metadata between tools and systems. According to (Marco, 2000), a metadata standard must be: (*i*) Technology-independent, in terms that it shall not be tied to a specific technology only available for a few; (*ii*) Vendor-neutral, since all interested vendors shall participate equally in its definition; (*iii*) Realistic in scope, as it shall limit its scope to a realistic ground, not covering everything; (*iv*) Widely implemented, if it is in fact implemented in tools of major software vendors. In a first stage metadata standardization efforts focused on tool interchanging with standards such as the CDIF (CASE Data Interchange Format) for interchanging of data between CASE tools or CWM (Common Warehouse Model) and OIM (Open Information Model) (Vetterli, Vaduva et al., 2001) for Data Warehousing tools. Other initiatives have been taken, such as the Dublin Core Metadata Initiative[1] whose major goal was the definition of standards for documentation management purposes.

However, for describing technical and domain details of Information Systems or Enterprises, the use of limited-scope metadata standards are not appropriate solutions. Most of the times the solution is to create custom standards. For that effect major software companies started developing general-purpose Metadata tools aiming at the enterprise-wide metadata management. These expensive "off-the-shelf" tools support extensibility natively and allow the representation of metadata in different abstraction levels, providing repositories and their correspondent management tools with advanced

[1] http://dublincore.org

Ferreira R. and Moura-Pires J. (2007).
EXTENSIBLE METADATA REPOSITORY FOR INFORMATION SYSTEMS AND ENTERPRISE APPLICATIONS.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - DISI*, pages 344-350
DOI: 10.5220/0002364503440350

metadata importation and exportation capabilities from and to other tools or repositories. As enterprise-wide tools these have advanced user access procedures, featuring user views for restricting access within metadata information, and highly complex architectures. However, due to their price and complexity these are not appropriate to medium size Information Systems with limited budget. For such cases, a metadata solution shall be low cost, light weighted, extensible and easy to integrate.

Since its appearance, XML[2] has been used in Information Systems for data and metadata interchanging, being adopted by most metadata standards. Several technologies were developed around XML for various purposes: (*i*) for validation, XML Schema[3] and Schematron[4]; (*ii*) for transformation and processing XSL[5], which allows the transformation of data in an input XML document into another format, such as XML or HTML; (*iii*) for querying and updating, XQuery[6] and Xupdate[7].

This paper presents a Metadata Repository solution, evolution of a previous work in the area (Ferreira, Moura-Pires et al., 2005), based on the use of XML and related technologies, providing means for an easy integration in existing Information Systems and featuring extensive metadata querying and transformation mechanisms for documentation generation.

The paper is structured as follows: this section presents the motivation, section two presents the design of the Metadata Repository, with section three presenting its architecture. Section four presents the implementation and section five a case study. Section six draws some conclusions and future work in this field.

2 REPOSITORY DESIGN

For the Metadata Repository design a set of requirements are considered. According to (Marco, 2000), the following data and operational requirements should be considered for the design of a Metadata Repository:

- Ability to handle metadata standards and custom metadata types;
- Ensure that metadata maintains its integrity and consistency while in the repository;
- Handle evolution of metadata through time by providing versioning and impact analysis;
- Support for various presentation styles providing advanced querying and transformation mechanisms for metadata;
- Support for Web-based formats such as HTML and XML;
- Provide simple interfaces for easy interoperability with other tools;
- Include means for recovery from error situations such as power or server failures, without affecting metadata and the operation of the systems that depend on it;
- Include access control and security mechanisms on metadata;
- Handle concurrency in distributed environments detecting and solve conflicting situations such as multiple users writing the same metadata;
- Provide metadata dependency navigation and analysis;
- Provide effective mechanisms for storing and finding metadata in the repository.

From our experience (Ferreira, Moura-Pires et al., 2005; Pantoquilho, Viana et al., 2005), a Metadata Repository should support all the stages of Information Systems' development lifecycle. A design option is to use XML and related technologies beyond data interchanging, but as the foundations for representing, storing, querying and transforming metadata information, in the repository. From the information point of view the Metadata Repository must be able to store and manage metadata information in various abstraction levels. A mapping of the information abstraction levels defined in the MetaObject Facility (MOF)[8] metadata standard, with the repository terminology and XML technologies is shown next:

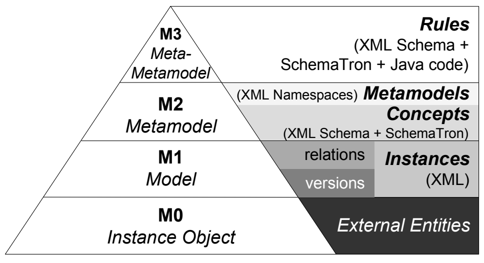

Figure 1: Metadata Information Model.

[2] http://www.w3.org/TR/2006/REC-xml-20060816
[3] http://www.w3.org/TR/xmlschema-0
[4] http://www.schematron.com
[5] http://www.w3.org/TR/xsl
[6] http://www.w3.org/TR/xquery
[7] http://xmldb-org.sourceforge.net/xupdate

[8] http://www.omg.org/mof/

Analysing the standard MOF model, on the left of Figure 1, the base of the hierarchy is the **M0** level. This level refers to source objects in a given reality, either being physical or non-physical entities such as persons, or information stored in a database. These objects are considered external, therefore are not included within the Metadata Repository.

The **M1** level (model) describes objects in the M0 level. A model is a finite description of a source object for a specific purpose. For example, the model of a conference paper can be composed of the paper title, abstract, conference name and authors. For dealing with change management requirements, a versioning system is proposed to allow storage of multiple versions of a model, and a model relationship mechanism for allowing the explicit establishment of relationships between models to represent their source object relations or other logical relations. In the Metadata Repository context, models are called **instances**, their versions are represented by **XML documents** and relations are established by using a pre-defined XML syntax.

The **M2** level (metamodel) describes the various models in the M1 level. A metamodel is the definition of a language to be used for a type of models, defining their fields and relations to other models. In the Metadata Repository, metamodels are called **concepts** and are defined using **XML Schema** and **Schematron** that are used for ensuring validity: all instances are checked if they are compliant with their concept definitions prior to their storage. By allowing custom definitions for concepts, the Metadata Repository is able to deal with any kind of metadata including metadata standards, a common requirement for Metadata Repositories (Marco, 2000). Furthermore, concepts can be grouped in **metamodels** according to the target **XML namespaces** defined in their schemas: two or more concepts that share the same target namespace belong to the same metamodel. The use of namespaces eases the integration of external metamodels, including metadata standards.

The **M3** level (meta-metamodel) defines descriptions for the various metamodels in the M2 level such as rules and common structure definitions for all metamodels. In the Metadata Repository these are called as **rules** and include properties and predefined structures to be used in every concept, with the purpose of enforcing concept structure consistency and standardization. Examples of those rules are: (*i*) common structures to be used in every concept (identification, versioning and authoring information); (*ii*) properties that every XML Schema must satisfy; (*iii*) syntax structures to declare and specify relationships between instances; (*iv*) checks to ensure instance and concept names uniqueness within the same metamodel; (*v*) syntax of internal global identifiers for instances and their versions. These rules are internal to the Metadata Repository, are implemented by **XML Schema**, **Schematron** and **Java** code, and are used for performing validity and support operations on concepts and instances.

Summarizing, instances are XML documents that follow the language defined by an XML Schema of a single concept; concepts can be grouped in metamodels by their schemas' target namespaces; and a set of design rules apply to all concepts. These rules include that every concept XML Schema and consequently every instance XML has in its root element attributes for storing its instance identifier and version identifier **(1)**, and elements for storing the instance name **(2),** version creation and modification dates **(3)**. This is shown in the following instance XML example:

```
<Paper globalId="1.1      1
      version="last"
      xmlns:mdr="http://di.fct.unl.pt/mdr">
2 <mdr:Name>Metadata Repository
for...</mdr:Name>
  ...
  <mdr:CreationDate>2006-12-
3 5</mdr:CreationDate>
  <mdr:ModificationDate>2007-03-
05</mdr:ModificationDate>
  ...
4 <Authors type="relation">
    <mdr:Relation name="Ricardo Ferreira"
semantic="paper author"/>                5
    <mdr:Relation name="João Moura-Pires"
semantic="paper author"/>
  </Authors>
</Paper>
```

Figure 2: Instance XML example.

Note in **(1)** the syntax of global identifiers for instances *(serverId . databaseId . instanceId)* and the syntax for version identifiers (numeric or *"last"* for the last version). The presented example shows an instance to model this paper, with its authors specified as relations to other instances **(4)**. The predefined syntax for referring related instances is presented in **(5)**, identifying the related instances by their name (that acts as identifier within the current namespace) and specifying a semantic for the relation using attributes. Other attributes exist for specifying the target namespace of the instance, which in this case was not used since the reference is within the same namespace, the identifier of the target instance version, which if omitted the last version is considered, and an XPath expression to refer to a fragment inside the target instance XML. Like all other instance elements, relation elements are declared in the concept XML Schema using a predefined complex type, and configured with an annotation where relation cardinality and eligible instance targets are defined by their concept names and namespaces.

3 REPOSITORY ARCHITECTURE

The Metadata Repository architecture is a service-based architecture that includes a logic engine running in a Web Service environment, accessible by an authenticated Web Service interface. This interface is used by all applications that interact with the repository, such as Metadata Importers (whose task is to import metadata from existing external sources), Management Tools (for managing repository configuration and metadata contents), and Information Systems or Applications (that manage their behaviour according to the metadata maintained by the repository). A management tool is provided with the repository for administering and managing its contents, allowing end users to upload, view, navigate and query metadata stored in the repository. Figure 3, shows the diagram of this architecture, and next paragraphs discuss each of the identified blocks inside the logic engine of the repository.

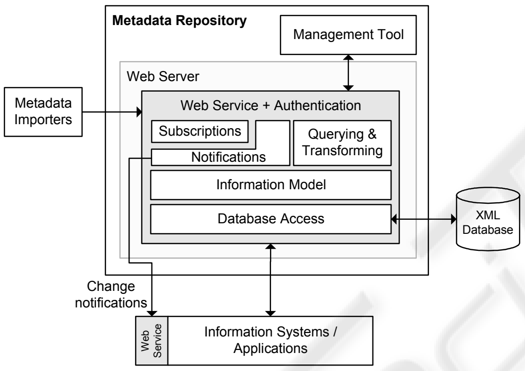

Figure 3: Metadata Repository architecture.

The Database Access block deals with all the accesses to the supporting databases, providing high-level functionalities to all the other blocks. Since the repository information model is based XML technologies, a XML database should be used. As requirements, this database must support XQuery, and preferably XUpdate technologies, and allow multiple instances running simultaneously, since each repository server can have more than one database. As functionalities this block shall interface with the database directly, managing database users and their access permissions on stored resources as well as providing a transaction management system for protecting all operations from logical or physical server failures. This last feature is one of the most important for ensuring the database integrity, and if the database does not implement it, this block shall ensure ACID (Silberschatz, Korth et al., 1997) on the database access, what can be achieved by using shared and exclusive locks acquired and released in a two-phase locking fashion, with deadlock prevention algorithms such as the Wound-Wait algorithm (Tanenbaum and Steen, 2002).

The Information Model is a core block that implements the model presented in the previous section. It is responsible for validating, storing and maintaining integrity of concepts, instances, and their versions. Instance versions are managed by this block, ensuring that each version has its version identifier, and that each instance has at least one version. Furthermore instance relations are also validated and maintained by this block, validating them in target and cardinality (according to their definition in the concept), and keeping their integrity once stored: the repository guarantees that the target of a relation always exists, by not allowing the removal of a referenced instance version.

The Querying and Transforming block provides advanced querying capabilities to the Repository, allowing the definition and execution of queries in XQuery language to be executed in the supporting XML database, to return concepts or instance versions. The XQuery language allows the definition of an output document that can be either an XML document or HTML, Text, or other type of document. If the query results are XML, they can be further transformed using a Single Transformation (XSLT) or a Transformation Pipeline (set of XSLT to be processed in pipeline). These are commonly used for generating HTML outputs from query results, offering documentation generation capabilities. The complete querying and transforming capabilities are depicted in Figure 4:

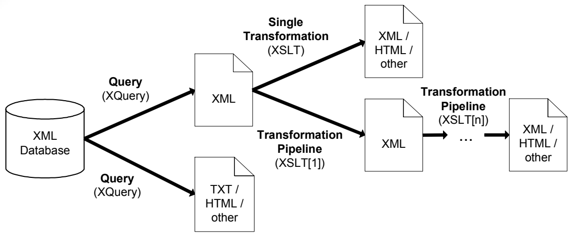

Figure 4: Complete querying and transforming capabilities.

The Notifications block allows the set-up of notifications from the repository to external web services to signal about instance modifications in a given repository database. These can be triggered once one of the following occurs: (*i*) a given set of instances are modified or removed; (*ii*) instances are added, modified or removed on a given set of concepts; (*iii*) instances selected by a given XQuery are modified or removed. Notifications help in integrating external systems and application with the repository, informing them when some changes occur on metadata needed for their operation.

The Subscriptions block allows subscribing instances from other repository databases of local or remote servers. The process of subscribing metadata from database A to database B is described as follows: (*i*) database A declares linked database B and sets the instances to share, that can be subscribed by database B; (*ii*) database B declares linked database A and chooses the instances to subscribe from the available. Once this process is completed and subscriptions processed, the subscribed instances will be made available in the repository database. These are treated differently from local instances, as presented next: (*i*) Subscribed instances are read-only and cannot be modified; (*ii*) Subscribed instances are updated automatically if modified in their source database; (*iii*) Relation integrity is not guaranteed for subscribed instances: the target instance might not be available if it was not subscribed; (*iv*) Local instances can relate with subscribed instances if specified in their concept definitions; (*v*) If a subscribed instance that being referred by a local instance is removed from the subscription, it will not be deleted from the Repository until the local instance keeps referencing it, therefore ensuring the restriction on local instance relations' integrity. Automatic update of subscribed instances is performed using notifications: once a subscription on a set of instances from a source database is established, a notification on that same set of instances is created on the source database, for invoking a method of the repository web service to refresh the remote database subscribed instances.

3.1 Fault Tolerance

From our view Information Systems and Applications shall model their behaviour and processes according to metadata stored and managed by the Metadata Repository. However if the repository is not available due to a fault, the operation of such systems and applications can be compromised or highly affected. Consequently fault tolerance mechanisms must be considered to ensure maximum availability of the repository.

To address availability, replication is a commonly used solution. In database systems several replication modes are available (Silberschatz, Korth et al., 1997), depending on data availability requirements and usage profiling. However the usage profiling of database systems is distinct from the usage profiling of a Metadata Repository: in a Metadata Repository writing operations are few, as opposed to reading and querying operations, that can be performed several times, and can involve large result sets. Therefore server replication can be simplified and targeted for reading and querying operations. Replication applies not only to the databases but also to the Metadata Repository server. For better results, multiple machines can be used, each one with a Web Server running Metadata Repository Web Service and supporting databases.

The proposed architecture for server replication is based on a Primary-Backup solution (Tanenbaum and Steen, 2002), where the primary server performs all the writing operations in first place, propagating them to the backup servers. The selection of the primary server in the group is performed automatically by using an election algorithm like the *Bully Algorithm* (Tanenbaum and Steen, 2002). For dealing with faults, all the servers know each other by being placed in a group, and if the primary server fails a backup server can take its place. Reading can be performed from any server, including the primary one (the most updated results are obtained by reading from the primary server, since write operations are propagated asynchronously to backup servers). This architecture is transparent for external applications, which communicate with the repository using a provided library. At start-up this library only needs to know the address of one of the servers in the group, after that it forwards the writing and reading operations to the available servers in the group.

4 IMPLEMENTATION

An implementation of the solution presented in this paper is currently running in operational state. The solution was implemented in the form of a Java Web Service (J2SE 5.0) running in Apache Tomcat 5.0. As supporting database, eXist Native XML Database[9] was used, since it is a Java-based open-source solution, with a wide community support, providing a large set of features: eXist includes an automatic index mechanism that indexes all the XML documents in proprietary data store, supports XQuery and XUpdate for querying and for updates in the databases, and provides various choices for integration, by its various interfaces. All presented functionalities were implemented with exception for Fault-tolerance, whose algorithms and techniques were simulated. The metadata subscription mechanism works by using Really Simple

[9] http://exist.sourceforge.net/

Syndication[10] (RSS) 2.0 feeds and web service calls between repository databases.

A Management Console was developed in form of a windows desktop application, developed in Microsoft C# and Visual Basic .NET 1.1. This management console features an easy and simple user interface, supporting multiple servers and databases, allowing visualization, navigation and management of all local and subscribed metadata, definition and testing of queries and transformations, notification management, definition of metadata shares and subscriptions. Screenshots and further information about the implementation is available in http://centria.fct.unl.pt/~jmp/mr.

5 CASE STUDY – SESS PROJECT

The Space Environment Support System for Telecom/Navigation Missions (SESS)[11], is a project sponsored by the European Space Agency which goal is to provide to spacecraft operators and scientists a set of tools to monitor and analyse information about the space environment and its effects on spacecrafts. This project is an evolution of a previous project in the same area, the Space Environment Information System (Pantoquilho, Viana et al., 2005).

The basic infrastructure architecture of the system is the following:

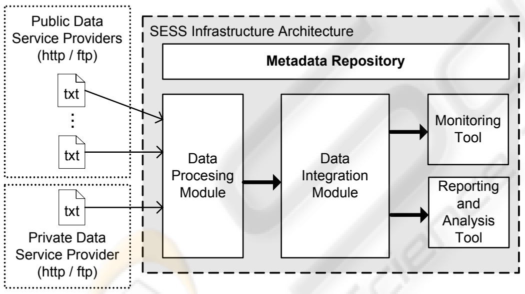
Public Data Service Providers (http / ftp)
txt
txt
txt
Private Data Service Provider (http / ftp)
SESS Infrastructure Architecture
Metadata Repository
Data Procesing Module
Data Integration Module
Monitoring Tool
Reporting and Analysis Tool

Figure 5: SESS infrastructure architecture.

Space Environment data is obtained from Public Data Service Providers (DSP), and proprietary data such as Spacecraft telemetry is obtained from Private DSP. This data is provided as semi-structured text files, human readable, containing tables and lists. The Data Processing Module (DPM) (Raminhos and Moura-Pires, 2007) downloads these files from their sources, extracts the relevant data from them (values), delivering it to the Data Integration Module (DIM). The DIM has a set of databases to store this data, namely an Operational Data Store, containing the most recent data, and a Data Warehouse, containing all historical data. Several data loading processes are included in DIM, for populating the databases automatically with data delivered by DPM. Two client tools are provided to the end-users, with distinct purposes: the Monitoring Tool provides near-realtime monitoring and alarming of space environment conditions, while the Reporting and Analysis Tool allows analysis of past data and creation of reports.

This system is rich in both domain and technical metadata. The description of each Space Environment measure (e.g.: radiation level, magnetic field intensity) and each spacecraft from which the measures are taken from, are examples of domain metadata. As for technical metadata some examples can be the Data Service Providers definitions, the files to download and their schedule, the format and subsequent transformation process to apply to these files and extract data, behaviour of database loading programs, database model diagrams, or global configurations of the system. As every component of the system is metadata-driven the metadata can be considered as the "glue" of the system. Therefore the Metadata Repository has an important role in the architecture, considered as the backbone of all the system, to which all the system components access. The Metadata Repository has around 30 concepts and 1400 instances.

The complete architecture of SESS was designed in accordance to a specific scalability requirement: the system must support multiple missions each having its private data. Therefore the architecture includes multiple Mission Infrastructures and a Common Infrastructure. The Common Infrastructure is controlled by the European Space Agency (ESA) and includes public space environment data that can be subscribed by any mission. Furthermore, a mission can also publish data from its Mission Infrastructure to the Common Infrastructure making it available for all missions. This architecture is shown next for ESA missions (XMM, ENVISAT and INTEGRAL):

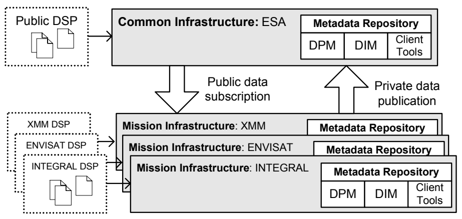

Figure 6: SESS Multi Mission Architecture.

[10] http://blogs.law.harvard.edu/tech/rss

[11] http://telecom.esa.int/telecom/www/object/index.cfm?fobjectid=20470

As explained before, each space environment measure stored in the database is described as instances in the Metadata Repository. Therefore, the data subscription mechanism is controlled by metadata subscriptions of such instances between infrastructure Metadata Repositories, using the process explained previously.

5.1 Documentation Generation

In the SESS project the Metadata Repository was also used as support tool for the automatic generation of management and technical documentation. One example is support to the analysis, requirements and testing phases of the project. In the first phases, the architecture of the system was discussed and meetings were held to define the user and system requirements of the system. By the definition of a set of concepts, analysis results and requirements were normalized, structured and saved in the Metadata Repository as instances. In the testing phase of the project, a concept for describing tests was defined, allowing the description of tests to be performed, their correlation to requirements and their results. Taking advantage of the instance versioning mechanism it was possible to evolve these instances during all project development phases, and thanks to the querying and transforming mechanisms, automatic documentation was generated, such as requirements tables or traceability matrices. The next diagram represents the concepts and their relationships for supporting these activities:

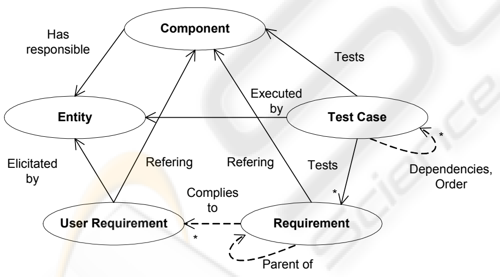

Figure 7: Concepts and relationships representation for supporting the analysis, requirements and testing phases.

6 CONCLUSIONS AND FUTURE WORK

The solution presented in this paper has proved by its implementation in the presented case study to be an appropriate answer for the integration and use of metadata in Information Systems or Enterprise Applications. The main advantages of the solution are its flexibility, lightness, ease of use and integration capabilities. In the near future, the presented implementation will be applied in other scenarios.

The use of XML technologies to represent, store, query and transform metadata information, proved to be most appropriate and pragmatic, given the maturity and wide availability of tools, parsers and databases, comparing to other formats for representing metadata such as the Resource Description Framework (RDF)[12] and RDF Schema[13] which have fewer tool support and immature technologies.

REFERENCES

Ferreira, R., J. Moura-Pires, et al. (2005). XML based Metadata Repository for Information Systems. 12th Portuguese Conference on Artificial Intelligence, Covilhã, Portugal.

Marco, D. (2000). Building and Managing the Meta Data Repository: A Full Lifecycle Guide.

Pantoquilho, M., N. Viana, et al. (2005). SEIS: A Decision Support System for Optimizing Spacecraft Operations Strategies. IEEE Aerospace Conference, Montana, USA.

Raminhos, R. and J. Moura-Pires (2007). Extraction and Transformation of Data from Semi-Structured Text Files using a Declarative Approach. ICEIS 2007 - 9th International Conference on Enterprise Information Systems, Funchal, Portugal.

Silberschatz, A., H. F. Korth, et al. (1997). Database System Concepts, McGraw-Hill.

Tanenbaum, A. S. and M. v. Steen (2002). Distributed Systems - Principles and Paradigms, Prentice-Hall.

Tannenbaum, A. (2002). Metadata Solutions – Using Metamodels, Repositories, XML and Enterprise Portals to Generate Information on Demand, Addison-Wesley Professional.

Vaduva, A. and K. R. Dittrich (2001). Metadata Management for Data Warehousing: Between Vision and Reality. 2001 International Database Engineering & Applications Symposium (IDEAS '01), Grenoble, France.

Vetterli, T., A. Vaduva, et al. (2001). Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metamodel. 2001 International Database Engineering & Applications Symposium (IDEAS '01), Grenoble, France.

[12] http://www.w3.org/RDF
[13] http://www.w3.org/TR/rdf-schema