XML and RDF Databases

Version 1.0

10.6.2003

Asta Bäck
Minna Suovirta
Sari Vainikainen
Samuli Viitanen
Version history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Reviewer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>27.5.2003</td>
<td>A.B., M.S., S.V., S.V.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Contact information

Asta Bäck
VTT Information Technology
P.O. Box 12041, FIN-02044 VTT, Finland
Street Address: Tekniikantie 4 B, Espoo
Tel. +358 9 4561, fax + 358 0 456 7052
Email: asta.back@vtt.fi
Web: http://www.vtt.fi/tte/
Abstract

In the internal VTT project "XML Databases" (XMLDB) we focused on XML databases and a RDF database. The main issue of the project was comparison of XML and RDF Schemas, and testing a native XML database (Xindice) and a RDF database (Sesame) in our case application. Also an overview of the XML support of the mainstream databases was made utilising published information.

XML support is already available for many mainstream databases, whereas RDF is only supported in some few databases that are of experimental nature. Even though RDF utilises the XML serialisation syntax, the data model is very different from the XML data model. The XML data model is a tree, and that of RDF a labelled directed graph. Therefore, the RDF information can be much better utilised when the tools understand its semantic structure.

Our case application dealt with managing and viewing user opinions. In addition to the opinions, some background information was collected from the users. In order to be able to better utilise other's opinions, it is useful to know some background information (metadata) of other users, or at least to know their opinions of many items, and in this way try to find other people who have similar preferences and interests. This metadata can be used to give explicit reviews or metadata, and indirectly e.g. within collaborative filtering applications.

We can expect metadata explosion on the net, because metadata vocabularies are being developed and utilised in the different areas, and metadata are an important in finding information or services. Therefore tools and solutions to store and utilise metadata will become important.

RDF is the language for describing metadata, and it can be used to manage distributed network like information better than basic XML. More RDF information can be added during the process, and the data can be navigated from many directions. RDF is also gaining importance as an ontology language for the Semantic Web applications. With ontologies the interoperability of different applications can be improved.

The Xindice XML database suits very well to research and prototype applications, because no schema is mandatory for the data. This means that the information content can be varied and modified during the development and pilot phase. This adds flexibility compared to utilising a database that requires a schema or a relational database.
Preface

This publication contains the results of the internal VTT project "XML Databases (XMLDB)". The aim of the project was to strengthen VTT's know-how in XML databases (natural and XML enabled) and take advantage of the XML database in processing and utilisation of XML files. During the project the focus was widened to include the comparison of XML and RDF Schemas, and utilising also an RDF database in a case application.

At same time as the XMLDB project, another internal VTT project XDAT (XML in Data Management and eCollaboration Architecture) was carried out. In it, practical tests were made of XML enabled databases. Therefore, the XMLDB project concentrated on making the case application with a native XML database and an RDF database.

The project group members that have contributed to this publication are Asta Bäck (sections 1, 4 and 6), Minna Suovirta (section 2), Sari Vainikainen (sections 3 and 5.3) and Samuli Viitanen (sections 5.1 and 5.2). In addition, Antti Pärnänen, Raimo Launonen and Caj Södergård have participated in the project.

Espoo 6.3.2003

Minna Suovirta
Editor
VTT Information Technology
Contents

Abstract.............................................................................................................................. i
Preface .............................................................................................................................. ii
Contents ............................................................................................................................. iii
List of Acronyms................................................................................................................. v
1 Introduction...................................................................................................................... 1
2 XML Support in Mainstream Database ....................................................................... 2
  2.1 Informix .................................................................................................................... 3
    2.1.1 Object Translator ............................................................................................. 3
    2.1.2 Web DataBlade .............................................................................................. 4
  2.2 SQL Server 2000 .................................................................................................... 5
  2.3 Oracle 9i .................................................................................................................. 6
3 RDF and XML ................................................................................................................. 8
  3.1 RDF and XML Data model .................................................................................... 8
  3.2 RDF Schema and XML Schema .......................................................................... 12
  3.3 Differences between XML and RDF ................................................................... 14
4 RDF Databases ............................................................................................................. 17
  4.1 Reasons for Special RDF Databases and RDF Query Languages ....................... 17
  4.2 Sesame ................................................................................................................... 18
    4.2.1 Sesame’s Architecture and Technical Features ............................................. 19
  4.3 Content Visualisation Utilising Ontologies ........................................................... 20
5 Case Application – User Reviews .............................................................................. 23
  5.1 XML Database ........................................................................................................ 23
    5.1.1 Adding Documents into the Database ............................................................ 24
    5.1.2 Differentiating Documents in the Same Collection ....................................... 25
  5.2 Visualising XML Data ............................................................................................ 25
    5.2.1 Clustering Documents with User Given Search Terms ................................ 26
    5.2.2 Showing out the Results ............................................................................... 27
    5.2.3 System Information ....................................................................................... 28
  5.3 Utilising RDF and an RDF Database ..................................................................... 28
    5.3.1 Evaluation Ontology ..................................................................................... 28
    5.3.2 Data Import ................................................................................................... 32
    5.3.3 Querying Possibilities of RDF Experiment ................................................... 33
    5.3.4 Conclusions of RDF Experiment .................................................................. 41
6 Conclusions.................................................................................................................... 42
References......................................................................................................................... 43
List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADO</td>
<td>ActiveX Data Objects</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BLOB</td>
<td>Binary Large Object</td>
</tr>
<tr>
<td>CLOB</td>
<td>Character Large Object</td>
</tr>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise Java Beans</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>Java RMI</td>
<td>Java Remote Method Invocation</td>
</tr>
<tr>
<td>JDCB</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual machine</td>
</tr>
<tr>
<td>ODCB</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>OLE DB</td>
<td>Object Linking and Embedding Database</td>
</tr>
<tr>
<td>OQL</td>
<td>Object Query Language</td>
</tr>
<tr>
<td>PL/SQL</td>
<td>Procedural Language extension to SQL</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDQL</td>
<td>RDF Data Query Language</td>
</tr>
<tr>
<td>RQL</td>
<td>RDF Query Language</td>
</tr>
<tr>
<td>SAIL</td>
<td>Storage And Interface Layer</td>
</tr>
<tr>
<td>SMIL</td>
<td>Synchronized Multimedia Integration Language</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifiers</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WebDAV</td>
<td>Web-based Distributed Authoring and Versioning</td>
</tr>
<tr>
<td>XDR</td>
<td>XML-Data Reduced</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XPath</td>
<td>XML Path Language</td>
</tr>
<tr>
<td>XQuery</td>
<td>XML Query Language</td>
</tr>
<tr>
<td>XSL</td>
<td>Extensible Stylesheet Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Stylesheet Language Transformation</td>
</tr>
</tbody>
</table>
1 Introduction

This report summarises the work that has been done in an internal VTT project XMLDB (XML databases). Initially, the project only wanted to assess the status of XML support in databases, but the focus was widened to include the comparison of XML and RDF Schemas, and utilising an RDF database in a case application.

RDF is the language for describing metadata, and it is also gaining support as an ontology language for the Semantic Web applications. Our case application dealt with managing and viewing user opinions. We can expect metadata explosion on the net, because metadata vocabularies are being developed and utilised in the different areas, and metadata are an important in finding information or services. Therefore tools and solutions to store and utilise metadata will become important.

User reviews are also metadata. They can be given as explicit reviews or metadata, and indirectly as is the case with collaborative filtering applications. Collaborative filtering is based on tracking user interactions (for example, items bought), and this information it then utilised to suggest new customers similar combinations of items.

Sometimes, the users might like to see more personal reviews. Travelling web sites are an example of this. In order to be able to better utilise other people's opinions, it would be useful to know some background information of other users, or at least to know their opinions of many items (or what ever that they are assessing), and this way try to find other people who have similar preferences and interests.

This report has the following structure: Chapter 2 includes an overview of the XML support that the mainstream database vendors currently offer according to the information that is publicly available about that. Chapter 3 presents and discusses the differences between RDF and XML. Chapter 4 discusses why specific RDF databases have been developed, and presents the Sesame database that was utilised in the project. Chapter 5 presents the XML and RDF applications that were developed in the project. The XML application emphasised data visualisation, and the RDF application, searching entries utilising the RDF data model. Chapter 6 presents the conclusions.
2 XML Support in Mainstream Database

XML-enabled databases are databases that contain extensions for transferring data between XML documents and themselves. XML-enabled databases are usually relational. Also all three commercial XML-enabled databases that will be concerned with in this document are relational (see Table 1) [Bou02a].

There are two primary methods for working with XML in relational databases: inbound and outbound translation and mapping of XML documents to relational data structures; or the storage of complete XML documents or fragments using pseudo-text structures such as BLOBs and CLOBs [Zap02].

Table 1. Databases with the XML support.

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
<th>XML Output</th>
<th>Select as XML</th>
<th>BLOB / CLOB</th>
<th>Text Search</th>
<th>XML SDK</th>
<th>Java Support</th>
<th>Price (one CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>Informix Object Translator</td>
<td>X</td>
<td>X</td>
<td>X¹</td>
<td>X</td>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>IBM</td>
<td>Informix Web Data Blade Module</td>
<td>X</td>
<td>X</td>
<td>X¹</td>
<td></td>
<td></td>
<td></td>
<td>10 091 $³</td>
</tr>
<tr>
<td>Microsoft</td>
<td>SQL Server 2000</td>
<td>X</td>
<td>X</td>
<td>X²</td>
<td>X</td>
<td>X</td>
<td></td>
<td>Standard 4 999 $⁴</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enterprise 19 999 $⁴</td>
</tr>
<tr>
<td>Oracle</td>
<td>Oracle 9i Release 2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>Standard 15 000 $⁵</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enterprise 40 000 $⁵</td>
</tr>
</tbody>
</table>

¹ IBM Informix Excalibur Text Search DataBlade Module [Inf02c] ² [Mic00] ³ [IBM03] ⁴ [MS02] ⁵ [Ora03]

The information presented in this report is based on information given by the respective database suppliers. In XDAT (XML in Data Management and eCollaboration Architecture), which was another internal VTT project carried out in 2002, practical tests were carried out with Oracle 9i and a native XML database, Xindice. The XDAT project research report [HMP03], also includes theoretical comparisons of the different types of databases and their suitability to saving XML data.
2.1 Informix

IBM Informix supports XML through its Object Translator tool and Web DataBlade module. The development of the Object Translator tool is finished [När02].

2.1.1 Object Translator

IBM Informix® Object Translator™ is a data integration tool that enables developers to create object models in Java, XML, and Visual Basic that represent an underlying database schema [Inf02a].

In the Object Translator is automated generation of XML-relational mappings. The Object Translator maps the elements/attributes (nodes) in XML document templates to the relational schema. The created 'map' object represents the object-relational mappings. The generated objects function as XML 'document handlers'.

Once the XML document template is mapped to a 'map' object, the Java object generated from the map object can save/restore the data in XML documents (that have the same structure as the mapped template) to/from the data in the relational columns. Markups (structure) of XML documents are not stored to the database.

The XML document to be mapped is opened inside a separate windowpane as a tree control. The user drags-drops elements/attributes to attributes of the 'map' object that already represent the underlying relational schema into which the XML document’s data is to be stored.

Object Translator supports only Java language based XML document handlers. Thus, when the user links XML document nodes to a ‘map’ object, they have to generate either EJBs or Java Beans from these map objects.

In the Object Translator it is allowed to map multiple XML document templates to the same Java object (EJB or bean) hierarchy. It can be multiple 'XML-centric views' of the information in the underlying database. This functionality allows for supporting multiple types of client views of the same data and is a requirement for all web applications/organisations that need to support different types of client devices at the same time and within the same application.

The Object Translator is able to generate XML document instances from DTDs or object schemas/models. In case of DTD, user can in a wizard-based user interface interactively choose among optional sub-elements that may comprise a XML document element. The user also gets to view the XML document tree as it is constructed. In case of object schemas, a wizard of the Object Translator takes as input an object hierarchy and outputs an XML document with the same structure. The generated XML document can be used as a template for the XML mapping process.

The Object Translator can convert HTML documents to XML documents. It makes HTML 'well-formed' and adds XML-specific tags. Once the XML has been generated, it can be used as input to the mapping process. Also this function is done with graphical user interface and wizards.
It is possible to automatically apply XSL stylesheets to incoming/outgoing XML streams. XSL stylesheets can be used to generate an HTML or a WML document from an XML document, convert an XML document based on one DTD (or XML Schema) to an XML document based on another DTD (or XML Schema) etc.

The Object Translator allows the user to generate SOAP (Simple Object Access Protocol) services on top of generated XML 'document handler' Java objects. The Object Translator SOAP wizard generates both a SOAP-enabled server that accepts SOAP message requests and a SOAP-compliant client application that can send appropriately formatted requests to the server. The SOAP servers can in turn talk to the Object Translator generated Java classes and retrieves data or sends data to them in XML format.

Object Translator also provides a simple mechanism for SOAP service 'discovery' on a specific web server, thus allowing the user the freedom of doing a discovery of services, followed by a binding to a specific service and then the invocation of the service. The advance of this functionality is that database access over the Internet can now be done in a standard-based manner.

With the Object Translator the user can make aggregation and personalisation of data based on the definition of custom 'web services'. In a 'Web Services' wizard the user can define a web service as an aggregation of information from multiple data sources. Object Translator assumes that each data source returns data either in XML format or as atomic text values. User can also customise access and retrieval of information from the disparate data sources.

2.1.2 Web DataBlade

The Informix® Web DataBlade module is a collection of tools, functions, and examples that ease development of "intelligent", interactive, web-enabled database applications. The web applications dynamically retrieve data from an Informix database. The Web DataBlade module supports most Web Server Application Programming Interfaces (APIs), and enables a truly interactive web site [Inf02c].

In the Web DataBlade module the user creates HTML pages that include Web DataBlade module specific tags and functions that dynamically execute the SQL statements, which the user specifies, and format the results. These pages are called Application Pages (AppPages). The types of retrieved data can include traditional data types, as well as HTML, image, audio, and video data.

AppPages are themselves stored in the database. A web application that uses the Web DataBlade module, therefore, first retrieves the AppPage from the database, then passes the AppPage through an SQL function that interprets the special AppPage tags and functions, typically to retrieve or update data from database tables and to format the results.

The MSQL tag generates XML-formatted data based on rows in a table. This can be done in two ways: by using the MSQL tag to retrieve only data from the table and specifying own XML tags, or by using XML-specific attributes of the MSQL tag to generate default XML tags as well as retrieve data from the table [Inf01].
2.2 SQL Server 2000

XML Support of the Microsoft® SQL Server™ 2000 covers the ability to access SQL Server over HTTP through a URL, support for XML-Data schemas and the ability to specify XPath queries against these schemas, and the ability to retrieve and write XML data. Some of XML support features are built into the server components, while other are provided on the mid-tier components accessible via ADO, OLEDB and HTTP. The server components include an XML Parser, an extension called 'FOR XML' to translate query results into an XML, and an XML rowset provider to convert XML documents into relational format [Bou02b, Nol00, Zap02].

Retrieving XML data is supported via the FOR XML clause of the SELECT statement. Three modes (RAW, AUTO, and EXPLICIT) are supported by FOR XML to provide different level of control over the generated XML.

Writing XML data is supported using the OpenXML rowset function. The OpenXML can be used as a table reference, which means that data of XML documents can be used for inserts or updates into database tables.

To use the OpenXML function the user first creates an internal representation of the XML document using a system stored procedure. This internal document is a tree representation of the various nodes of the XML document. A XPath row pattern defines nodes of the XML documents to process. By default, XML attributes are used in the mapping between the XML data and the relational rowset, but user can also define mapping to be element-centric.

The basic format of schema declarations is similar to table declarations. The main inclusion would be an optional XPath column pattern.

SQL Server also provides support for XML hierarchy and overflow handling that allows XML data not mapped directly to relational data to be placed into additional columns.

Annotated schemas provide a fully bi-directional XML-based programming model for querying and updating data. Microsoft's XDR and XML Schema formats are annotated with mapping information to the relational tables and views that can be queried using XPath. In the future, Microsoft will supports XQuery for these queries.

With ActiveX Data Objects (ADO) and Visual Basic it is easy to extract SQL Server data in XML format for components. The key of a code to perform this operation is the ADO Stream object in ADO 2.6, which lets the user retrieve XML into a Stream object. The XML data is returned to this Stream object in text format. The Stream can also contain relational data that will be transformed to XML by FOR XML clause. This means that the data does not necessarily have to be from SQL Server; it can be from any data store that can be accessed via OLE DB, including OLAP services.
2.3 Oracle 9i

Oracle 9i XDB supports both XML-enabled and native storage of XML data. It provides SQL features that allow users to view relational data as XML and XML data as relational. It fully absorbs the W3C XML data model into the Oracle server, and provides new standard access methods for navigating and querying XML. The Oracle 9i DXB has advantages of relational database technology and XML technology at the same time. XDB also adds an XML Repository to the database [Bou02b, Ora02, Zap02].

The main feature of the Oracle 9i is the XMLType data type, a new system defined object type that includes number of useful methods to operate (create, extract and index) on XML data. The XMLType enables SQL operations on XML content as well as XML operations on SQL content. Like any object type, XMLType can be used as the data type of a column in a table or view, as well as in PL/SQL stored procedures as parameters and return values. An XML view can be constructed over any data, regardless of whether it is relational data or XML data.

The XML data type provides lot of features for managing XML storage, such as XML Schema support, XML Schema storage with DOM fidelity, XML piecewise update, XPath search, XML indexes, XML operators, XSL transformations for XMLType, lazy XML load, XML views, Java Beans interface, and Schema caching.

The XML Schema support means that tables and types are automatically created, when the user has given a XML Schema extending the normal SQL DDL (Data Definition Language). This means the user has a standard data model for all her data (structured and unstructured), and can use the database to enforce this data model.

The DOM fidelity within the XML Schema storage means that user's programs can manipulate exactly the same XML data that she got.

Individual XML elements(s) and attribute(s) of documents can be updated piecewise without rewriting the entire document by using XPath to specify individual element(s) and attribute(s).

Via XPath the user can specify elements to query, and then use SQL operators on these elements. The XPath is also used to specify parts of a document to create XML indexes for XPath searches. In that way the user gets fast access to XML documents via XPath.

XSLT can be used to transform XML documents via a SQL operator for XMLType.

XMLType provides a virtual DOM. It only loads rows of data as they are requested (lazy XML load), throwing away previously referenced sections of the document if memory usage grows too large. This gets high scalability when many concurrent users are dealing with large XML documents.

The user can create XML views to construct permanent aggregations of various XML document fragments or relational tables.
XML DB provides a *Java Bean interface* for fast access to structured XML data, and extensions that save only those parts that have been modified in memory. The user can get static access as well as dynamic access to XML.

XML DB keeps structural information (like element tags, data types, and storage location) in a *schema cache*, to minimise access time and storage costs. Due to this the Oracle 9i has high performance and scalability.

XMLType data can be stored in either of two ways: with object-relational storage or as a CLOB. The storage options are interchangeable and XML applications use the same code regardless of which option is chosen. Changing from one storage type to the other requires a database export and import.

When using object-relational storage, users define the mapping with an annotated XML Schema. This states the name and data type (which can be an object type) of the SQL structure used to store a given element or attribute. Users can create their own mappings or use a default mapping generated by XDB. The XDB can round-trip XML documents at the level of the DOM. To do this, it uses hidden columns to store information that is not directly modelled by SQL, such as sibling order, processing instructions, comments, and whether a column corresponds to an element or an attribute. This information can be accessed via APIs in SQL or Java.

There are a number of practical differences between object-relational and CLOB storage. CLOB storage can round-trip XML documents exactly and so maintain accuracy to the original XML, while object-relational storage can round-trip them at the level of the DOM. CLOB storage uses text indexes, while object-relational storage uses BTree indexes. Data stored with the object-relational mapping is directly available to non-XML applications, while data stored with CLOB storage can only be accessed by XML-aware applications.

XMLType data can be accessed in several ways. Java Beans (which can be generated from an XML Schema) can be used when the data uses object-relational storage. The DOM can be used regardless of the storage option. Both methods can cache changes and store them later. In addition, data can be accessed executing SQL statements.

The other major feature of XDB is the XML Repository, which provides a file system-like view of XMLType objects in the database. XMLType objects (regardless of whether they actually contain XML data or are just XML views over relational data) can be assigned a path and corresponding URL in the repository hierarchy. These can then be accessed via WebDAV, FTP, JNDI, and SQL. In addition, the repository maintains properties for each object, such as owner, modification date, version, and access control.
3 RDF and XML

3.1 RDF and XML Data model

XML (Extensible Markup Language) and RDF (Resource Description Framework) are current standards for establishing semantic interoperability on the web.

XML addresses document structure and it can be utilised for several purposes. XML can be used for semantic markup of web pages. An XML serialisation can be used in a web page with XSL style sheet to render the different elements appropriately. XML is widely used as uniform data-exchange format. An XML serialisation can be transferred as a data object between two applications. XML is also used for serialisation syntax for other markup languages like SMIL (The Synchronized Multimedia Integration Language) [Dec00].

RDF is a general format for metadata encoding. The idea of RDF is to use metadata to describe the data contained on the web in order to make data machine-understandable. The huge amount of data on the web makes it difficult to find, present and maintain the information. It is very hard to automate anything on the web. The semantics of data helps computers for better searches and reasonings. RDF defines the rules for defining and expressing the semantics. Each author is free to define his own semantics and vocabulary. The schema which declares the semantic is expressed in the URI, which is used to declare the element names. RDF provides a syntax so that independent parties can exchange it and use it.

XML itself is not semantic. XML allows authors to create their own markup (e.g. <AUTHOR>), which seems to carry some semantics. XML may help humans to predict what information lie "between the tags", however, from a computational perspective a tag like <AUTHOR> carries as much semantics as a tag <H1>. A computer simply does not know, what an author is and for example, how the concept author is related to the concept "person". RDF expresses how different elements relate to each other [Cov98].

A well-formed XML document creates a balanced tree of nested sets of open and close tags, each of which can include several attribute-value pairs. The basic XML data model is a labelled tree, where each tag corresponds to a labelled node in the model, and each nested sub-tag is a child in the tree. In Figure 1 is shown a part of XML instance data and related XML Schema.
Figure 1. An excerpt of our XML Schema and related instance XML data of our seminar evaluation case.
The basic RDF data model consists of three object types describing the types of triples: resources, properties and statements. RDF properties may be thought of as attributes of resources and in this sense they correspond to traditional attribute-value pairs. RDF properties also represent relationships between resources and a RDF model can therefore resemble an entity-relationship diagram [Las99].

The basic building block in RDF can be expressed as a subject-predicate-object triple, which can be written as predicate(subject, object). A RDF statement can also be diagrammed pictorially using directed labelled graphs (also called "nodes and arcs diagrams"). In these diagrams, the nodes (drawn as ovals) represent resources and arcs represent named properties. Nodes that represent string literals will be drawn as rectangles [Las99]. A part of RDF Data is shown as a directed label graph, in text format and as subject-predicate-object triples in Figure 2. The data is taken out of our case, which is explained more in detailed in Chapter "Case RDF".

The big difference between XML and RDF is that XML creates a labelled tree and RDF a labelled directed graph. The ordering and nesting of elements is often significant, which makes it hard to express multidimensional relations in the XML serialisation, even though it is no problem to create n-dimensional graphs [Hje01]. The XML data model is strong in capturing positional collections of data since the children of an XML element are textually ordered. It is weak in capturing non-positional collections since these would suggest arc labels, absent in XML, which indicate the role each of the unordered components is playing in the collection [Bol01].
RDF Data

RDF Schema

- Employee
  - typeOf (instance)

- WholeSeminar
  - typeOf (instance)

RDF Data

Subject Predicate Object

- evaluation2_00121 evaluateWholeSeminar evaluation2_00122
- evaluation2_00121 age 36-40
- evaluation2_00122 grade 3

Same data expressed as subject-predicate-object triple

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>evaluation2_00121</td>
<td>evaluateWholeSeminar</td>
<td>evaluation2_00122</td>
</tr>
<tr>
<td>evaluation2_00121</td>
<td>age</td>
<td>36-40</td>
</tr>
<tr>
<td>evaluation2_00122</td>
<td>grade</td>
<td>3</td>
</tr>
</tbody>
</table>

predicate(subject,object)

evaluateWholeSeminar(evaluation2_00121,evaluation2_00122)
age(evaluation2_00121,evaluation2_00121,36-40)
grade(evaluation2_00122,3)

Figure 2. Example of RDF Data as a directed label graph, in text format and as subject-predicate-object triples.
### 3.2 RDF Schema and XML Schema

The XML Schema is used to define the structure of the information in a whole class of XML documents. It describes the possible arrangement of elements and content. In the XML Schema, two kinds of constraints can be used to create the model of the document infrastructure. These are the content model constraints, which describe the order and sequence of elements and a data type constraints that describes what constitutes valid units of data.

In the XML Schema, it is possible to define data types for the elements in the document, which is a major difference compared to the RDF Schema. The XML Schema contains simple types (e.g. ENTITY, NOTATION, String, Int, Boolean etc.) and complex types. The complex types allow elements in their content and may carry attributes; the simple types cannot have element content or attributes. An example of the XML Schema is shown in Figure 1. The original elements can be constrained further in order to create new elements. For example MaxExclusive and MinExclusive data types are used for setting boundaries in Schemas and Length, Enumeration, Literal are examples of data types that determine the numeric and literal type of an element.

Just as an XML Schema provides a vocabulary-definition facility for XML data, a RDF Schema lets developers define a particular vocabulary for RDF data. The RDF Schema defines the terms that will be used in RDF statements and gives specific meaning to them. The idea is that terms should hold the same meaning for all users. RDF Schemas define the meaning, characteristics, and relationships for a set of properties. When constructing the RDF Schemas new vocabularies are introduced through XML namespaces.

The RDF Schemas declare not only the syntax of a document, but they also provide mechanism for defining the relationships between the properties and other resources. In the XML Schemas, the content of the elements, the attributes, and the ordering and structuring of the elements (e.g., which elements can be nested in each other) are described for a given document type. But there is nothing that describes the relation between different resources. That is what RDF does.

The RDF Schema has three core classes that are used to define the elements under discussion: Resource, Property and Class. The RDF Schema makes it possible to indicate that certain classes are subclasses of others, and it provides a small number of basic classes. Resources may be instances of a class. This is indicated using the `type` property. Classes and subclasses are often organised as hierarchies, and the `SubClassOf` property describes the relationship between the classes and the subclasses. `SubPropertyOf` does the same for properties. There is a constraint mechanism in RDF defining two properties: range (which implies that the value of a property should be a resource of a certain class—in essence, a URI) and domain (which implies that the property can only be used on resources of a certain class). [Hje01] A part of RDF Schema is shown as graph and text format in Figure 3. The Schema was defined for our evaluation case, and more information about the RDF case can be found in Chapter "Utilising RDF and an RDF database".

In RDF, there is only one data type: strings. More type information can be added at the schema level by introducing constraints, but there are no data types in the RDF Schema. The information about data types is found in the XML Schema, therefore RDF should reuse the data types of the XML Schema [Hje01].
Figure 3. The part of RDF Schema as graph and text format. To keep picture simple all text elements are not drawn into the graph.
3.3 Differences between XML and RDF

The relationship between XML and RDF Schemas has been the root of a lot of confusion. The XML Schema describes the document in terms of basic data types. The RDF Schema defines the properties at application level and not only the properties of the resource (Title, Author, Subject, Size, Color, etc.) but also what kind of resources are being described (books, web pages, people, companies, etc.). The role of the RDF Schema is to provide a mechanism for declaring the properties of elements as well as relationships between properties and resources, and the semantics for these [Hje01].

The web is many-to-many data-interchange medium and it poses new requirements for any exchange format. One of the most important requirements for an exchange format is that data be understandable. Semantic interoperability is about defining mappings between terms within the data, which requires content analysis. Syntactic interoperability, which is about parsing data, is important factor, too. Applications must be able to read the data and get a representation that can be exploited.

XML has disadvantages when it comes to semantic interoperability. XML's major limitation is that it just describes grammars. There is no way to recognise a semantic unit from a particular domain because XML aims at document structure and imposes no common interpretation of the data contained in the document.

XML is useful for data interchange between applications that both know what the data is, but not for situations where new communications partners are frequently added. One domain model cannot be mapped easily to another because they are both encoded in schemas. To change XML documents, the domain mapping must be translated using mapping procedures such as XSL Transformations (XSLT) for grammars.

When it comes to semantic interoperability, RDF has significant advantages over XML: The object-attribute structure provides natural semantic units because all objects are independent entities [Dec00].

As a summary of comparison of the two schema languages [Hun01]:

- The RDF Schema provides support for rich semantic definitions through its ability to define type hierarchies, class/property relationships and human-readable descriptions (using the label and comment tags). But it provides limited support for the specification of local usage constraints (i.e. structural, cardinality and datatyping constraints).
- The XML Schema provides support for explicit structural, cardinality and datatyping constraints but provides little semantic information. The semantic knowledge is necessary to enable flexible dynamic mapping between metadata domains.

When comparing elements age and grade in XML Schema (Figure 1) and same properties in the RDF Schema (Figure 3) the differences between expressing the data types can be seen. The evaluation-RDF Schema is made with Protégé-2000 ontology editor tool and the range-constraint is determined as Protégé "Symbol"-type with the allowed values. The data types are expressed using namespace "http://protege.stanford.edu/system#".

In our case, the difference between the RDF Schema and XML Schema is that the RDF Schema also includes information about the actual seminar place, lecturers and programs.
This makes it possible to include information (like name, topic and contact information of lecturer) together with evaluation results. The XML Schema contains no data about the seminar itself. This information could have been included by defining an XML Schema for the seminar information, and either linking it to the evaluations, or by writing the program logic for connecting evaluations to the correct seminar. Nevertheless, this is more straightforward to solve with RDF than with XML. The RDF data file itself contains the information about the actual seminar. The evaluation data can easily be linked with the seminar information utilising properties. See Figure 4.

There is overlap in functionality in the following areas [Hun01]:

- Between the RDF Schema range constraint and the XML Schema type constraints.
- Between the RDF Schema domain constraint and the content model definitions of the XML Schema types and elements.
- In the definition of the enumerated list or controlled vocabulary values for organisations.
- Between the RDF Schema comments and the XML Schema annotations. Both provide human-readable descriptions of metadata elements or types.

The XML and RDF data models have complementary strengths and weaknesses. Several attempts have been made to somehow merge together the XML and RDF data models. More information can be read from articles [Bol01, Mel99, Pat02]. Depending on the solution, some changes are required to XML, to RDF or to them both. Approach for combining XML Schemas and RDF Schemas is dealt in article [Hun01]. In the article is proposed a web metadata architecture, which combines the best features of both the XML Schema and the RDF Schema to enhance metadata interoperability across the web.
Figure 4. The RDF Schema in our evaluation case includes also information about the seminar itself. Here is an example of RDF data and its connections to RDF Schema about the data of seminar place. All the data in text format is not drawn into the picture.
4 RDF Databases

4.1 Reasons for Special RDF Databases and RDF Query Languages

Resource Description Framework, as its name implies, is a framework for describing and interchanging metadata. It is built on the following rules [Bra01]:

1. A **Resource** is anything that can have a URI; this includes all Web pages, as well as individual elements of an XML document. An example of a resource is a draft of the referenced document [Bra01], whose URL is http://www.textuality.com/RDF/Why.html

2. A **Property** is a Resource that has a name and can be used as a property, for example Author or Title. In many cases, all we really care about is the name; but a Property needs to be a resource so that it can have its own properties.

3. A **Statement** consists of the combination of a Resource, a Property, and a value. These parts are known as the 'subject', 'predicate' and 'object' of a Statement. An example Statement is "The Author of http://www.textuality.com/RDF/Why.html is Tim Bray." The value can just be a string, for example "Tim Bray" in the previous example, or it can be another resource, for example "The Home-Page of http://www.textuality.com/RDF/Why.html is http://www.textuality.com."

4. There is a straightforward method for expressing these abstract Properties in XML, for example:

```xml
<rdf:Description about='http://www.textuality.com/RDF/Why-RDF.html'>
  <Author>Tim Bray</Author>
  <Home-Page rdf:resource='http://www.textuality.com' />
</rdf:Description>
```

As the previous example shows, RDF can be expressed in XML. However, XML databases are not very well suited to storing and querying RDF because the RDF data model is very different from the XML data model. XML documents are hierarchical, whereas RDF data model consists of triplets, or directed graphs. When making RDF queries, the relations are often of interest and queried, and not only the structure.

Different options to make queries are discussed in [BrK01]. RDF documents and RDF Schemas can be considered to be queried at three different levels of abstraction:

1. at the syntactic level, when they are treated as XML documents
2. at the structure level, when they are seen as consisting of a set of triples
3. at the semantic level, when they constitute one or more graphs with partially predefined semantics.

In an XML query language such as XQuery, expressions to traverse the data structure are tailored towards traversing a node-labelled tree. However, the RDF data model is
typically a graph, not a tree, and moreover, both its edges (properties) and its nodes (subjects/objects) are labelled.

As structure level, any RDF document represents a set of triples, each triple representing a statement of the form Object-Attribute-Value. A number of query languages have been proposed and implemented that regard RDF documents as such a set of triples, and that allow to query such a triple set in various ways. A major shortcoming in any query language that treats RDF documents only as triples, is that they will return in queries like, which resources are known to be of type FamousWriter, only the explicit triples, and does not utilise the information that is stored in the object hierarchy.

In the OnToKnowledge project, the conclusion was that RDF(S) documents should really be queried at the semantic level, which meant that all the information that is included in the RDF documents will also be utilised in queries. RQL is such a query language. According to [BrK01], RQL was developed within the European IST project C-Web and its follow-up project MESMUSES by the Institute of Computer Science at FORTH, in Greece.

RQL is a typed, declarative query language, and it adopts the syntax of OQL (Object Query Language), standardised by ODMG. As OQL, RQL is a functional language: the output of RDF Schema queries is again legal RDF code, which allows the output of queries to function as input for subsequent queries.

RQL is defined by means of a set of core queries, a set of basic filters, and a way to build new queries through functional composition and iterators. The core queries are the basic building blocks of RQL, which give access to the RDF Schema specific contents of an RDF triple store, with queries such as Class (retrieving all classes), Property (retrieving all properties) or Writer (returning all instances of the class with name Writer). This last query returns of course also all instances of subclasses of Writer, since these are also instances of the class Writer, by virtue of the semantics of RDF Schema. RQL can also query the structure of the subclass hierarchy. RQL also allows a select-from-where construct. RQL has path expressions, which allow matching patterns along entire paths in RDF/RDF Schema graphs.

There are several query languages and tools for RDF, or RDF based ontologies, and a recent survey of them can be found in [Mag02].

4.2 Sesame

The XMLdb project did some practical tests with the Sesame, a RDF Schema-based Repository and Querying facility. It was originally developed by Aidministrator Nederland bv as a research prototype and one of the key deliverables in the previously mentioned On-To-Knowledge project (EU-IST-1999-10132). Later, the OntoText project joined the On-To-Knowledge project and committed some resources to extending Sesame with versioning and fine-grained security functionality [Anon02].

After the On-To-Knowledge project, the NLnet Foundation committed to sponsoring further development under the condition that the software would be released as Open Source. Now, Sesame is available as an Open Source tool under the terms of the GNU Lesser General Public License (LGPL). Also, OntoText will keep supporting the project by investing more resources into its development.
4.2.1 Sesame's Architecture and Technical Features

The main feature of Sesame's architecture is that the actual storage of RDF, functional modules offering operations on this RDF, and communication with these functional modules from outside the Sesame system are clearly separated from each other.

![Sesame Architecture Diagram](image)

**Figure 5. Sesame architecture [Raa01]**

A generic API for RDF (Schema) repositories known as RDF SAIL (Storage And Inference Layer) is used to abstract from the actual storage device. This API has methods for storing, removing and querying RDF in/from a repository. Implementations of this API are used by Sesame's functional modules, but can also be used by stand-alone tools that need some kind of repository for storing their RDF.

It is possible to build a SAIL implementation on top of any kind of storage; be it a database, a file, or a peer-to-peer network. Current implementations are based on relational databases and Java graph models (memory storage) using plain files for persistent storage.

There are currently six functional modules in the Sesame system:

- A data administration module for adding and removing data.
- A RDF Export module that can be used to export (parts of) the data in a repository as a RDF document.
- An RQL query engine that can be used to evaluate RQL queries.
• An RDQL query engine that can be used to evaluate RDQL queries. (Not shown in Figure above.)
• A security module for more fine-grained security mechanisms. (Not shown in Figure above.)
• A versioning module for ontology versioning. (Not shown in Figure above.)

Protocol handlers handle communication with the functional modules from outside the Sesame system. There are protocol handlers that handle communication over HTTP, Java RMI and SOAP.

There is storage support for PostgreSQL, MySQL and Oracle databases.

One Sesame server can handle multiple repositories. A server offers the following functionality:
• Data uploads: the server can handle RDF data in XML-serialised RDF and N-Triples formats.
• Data extraction: the server can extract the contents of a Sesame repository in XML-serialised RDF, N-Triples, and N3 format.
• Querying: the server can evaluate RQL and RDQL queries.
• Data removal: the server can delete individual (or sets of) statements from a repository.
• RDF Explorer: a web-based browsing tool for exploring the contents of a Sesame repository.

Sesame offers programming interfaces on three different levels:
• The client API for client-server computing. This is the most flexible method of programming.
• The server API. This allows you to use all of the server features without the communication overhead.
• The lower level Storage And Inference Layer (SAIL), a generic API for RDF repositories. This will only offer basic methods for storing and retrieving RDF data and schema statements from a repository.

4.3 Content Visualisation Utilising Ontologies

Visualisation utilising information stored with connection to an ontology is presented in [FSH02]. One example is cluster mapping where instances are placed so those similar or partly similar instances are placed closer to each other. Clusters that belong to several classes are shown with connecting lines (See Figure 6).
Figure 7 shows another way of utilising the ontology. The users of a travelling eCommerce site can view the available choices by navigating through it in different order. At top level, the navigation can be started by defining either the type of accommodation, country or number of people travelling. Based on these, additional criteria are presented in a stepwise manner.
Figure 7. Ontologies are utilised in an eCommerce web site to help users in navigating in different order, and after each criteria selection, the items are shown, and the user may refine the search further. In the upper image, the country was selected first, and in the lower one, the type of accommodation.
5 Case Application – User Reviews

In the XMLDB-project, a survey was made about the seminars that the different research areas organise. Those who attended the seminar were asked to review the seminar as a whole, and also to make evaluations of its different aspects.

The idea with the case application was to support sharing individual opinions and to support individual decision-making. The application is not meant to be used as a statistical tool to analyse a great mass of evaluations.

The following evaluations were made:

- The seminar as a whole
- How interesting the program seemed beforehand
- How interesting the program really turned out to be
- Outside trainer or lecturer
- Leisure activities
- Food
- Accommodation
- Lecture room
- The seminar location and its surroundings

The scale from 1 (very poor) and 5 (excellent) was used. The respondents were also given a change to give free text comments of each aspect.

Each respondent was also asked to give the following background information of him or her:

- Age (5 year intervals)
- Gender
- Years at VTT (intervals with varying length)
- Education (student, masters, doctors, other)
- Activity in attending work related events (scale of 5) and
- Research area where he or she belongs.

5.1 XML Database

The XML database used in this project was Xindice [Xin02]. Xindice is a quite simply implementation of a native XML database engine that is entirely written in Java. As such it must always be hosted by Java Virtual Machine (JVM). Xindice implements the XML:DB API [XML02]. The XML:DB API is conceptually similar to ODBC and JDBC, but it has different levels of interoperability.

Logically, all XML data stored in Xindice is organised into a hierarchy of Collections. A collection is exactly what its name suggests: it contains any number of XML documents, and can in addition contain its own child collections, thus providing a hierarchy. The root
collection named the Database is the top of the hierarchy. It has no parent, and it can not contain XML documents of its own, only child collections.

Xindice organises XML data without XML Schema and it only requires that the documents to be added into database be well formed. This feature makes Xindice very flexible and suitable for the experiments like this.

5.1.1 Adding Documents into the Database

Addition of documents was implemented with the help of a web form (see Figure 8) and Java servlet technology [Sun03]. People first responded into a feedback form in the web.

![Figure 8: The feedback form](image-url)
The form then sends an HTTP POST request to a special Java servlet, which acts as the mediator - the servlet creates XML from the feedback data and then sends it forward to the database.

### 5.1.2 Differentiating Documents in the Same Collection

While added into database, each document is given a unique identifier value (stored into the XML hierarchy as an attribute of the root element). With this value the documents can be differentiated later from each other.

### 5.2 Visualising XML Data

To be precise the idea wasn’t to visualise XML data itself, but rather to visualise the similarity clusters of several XML documents. Two different kind of methods were created to do this:

**Method 1. Comparing the structure of the documents (DTDs)**

This method doesn’t look into the content of the element. It simply checks for the same element names. For example the element:

```xml
<item><gender>man</gender></item>
```

in document 1 is equal in this comparison to the element:

```xml
<item><gender>woman</gender></item>
```

in document 2. This kind of comparison might be useful when having piles of unclassified XML data. The program can separate in a matter of seconds from thousands of documents (depending a bit on the size of the documents), which documents include product data, which customer data etc.

**Method 2. Comparing the content of the documents (the actual data)**

In this case both the path (```<item><gender>``` ) and content (man/woman) must be equal. In the use case implemented in this project all the documents were formed with the same DTD and therefore this method was used to compare the data.

**Counting the clusters**

The principle with which the clusters were counted is known as Graph-Theoretic Clustering [Zah71]. Each XML-document is described as one node in a graph, which has an edge with every other node. The distance value between two nodes describes the similarity of these documents – the smaller the distance, the smaller the difference. After the graph has been formed, a minimal spanning tree is counted from it using Prim’s algorithm [Sas03]. Prim was selected because it is known to be effective for dense graphs like in this case. After the MST is counted the edges with largest lengths are deleted from it, leaving the final clusters.
5.2.1 Clustering Documents with User Given Search Terms

A user first gives the terms, with which she wants to differentiate the documents. This is done with a web form (Figure 9).

![Figure 9: Giving differentiation factors](image)

The terms are sent to a Java servlet, which then connects to the database, compares all the documents in the collection against the given terms, and counts the result clusters according them. The result is given to the user as an SVG [SVG01] picture (Figure 10).
In the result picture each square represents a link to one XML document and clusters are differentiated with colour and location.

5.2.2 Showing out the Results

Counting the XML clusters and their distances from other clusters was relatively easy. The difficult part was to show out the results in an understandable way to the user. If clustering is done with one differentiating factor (e.g. area), the clusters are easy to understand – in the picture above the documents in yellow cluster might be from TTE1, the ones in green cluster from TTE2 etc. If clustering is done with two factors (e.g. area and gender), the result picture becomes much more complicated to understand.

The conclusion was that even if the program counts a proper result with no matter how many differentiating factors, it is better to show pictures with at maximum three factors. Documents separated by three factors can be presented understandably in the picture: First the documents are differentiated by factor 1 to their primary clusters. Then by factor 2, which gives the right colour, and finally by factor 3, which divides documents into secondary (smaller) clusters inside the primary ones.
5.2.3 System Information

<table>
<thead>
<tr>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Operating system</td>
</tr>
<tr>
<td>Java editions</td>
</tr>
<tr>
<td>Compiler</td>
</tr>
<tr>
<td>XML database</td>
</tr>
<tr>
<td>Installed services</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser</td>
</tr>
<tr>
<td>Translator1</td>
</tr>
</tbody>
</table>

5.3 Utilising RDF and an RDF Database

5.3.1 Evaluation Ontology

The difference between the XML Schema and RDF Schema is that the RDF Schema includes also the information about the actual seminar place, lecturers and programs. This makes it possible to make queries starting from these pieces of information and to include information (like the name, topic and contact information of lecturer) with evaluation results.

The ontology was created with the Protégé-2000 ontology editor, which is developed at Stanford University (http://protege.stanford.edu/index.html).

The evaluation ontology includes six main classes (see Figure 11): Employee, who is the person who has taken part in a certain seminar and gives the evaluation from it. Instances of employee-class contain also background information (like age, education, and experience) about the person. The Evaluation-class contains ten subClasses. These subClasses are things to be evaluated like the accommodation and content of seminar. Instances of these classes contain the evaluations of a certain topic. Organisation-class includes information about the organisation that has arranged the seminar. Other classes

1 It is also possible to run program from command lines, not with a browser
contain information about the seminar itself. The instances of the program-class contain
information about the program topics (both free time and official lecture parts) of the
seminar. The instances of the seminar-class contain information about which organisation
arranged the seminar, the seminar program, and where the seminar was held. The
ServiceSupplier class has three subClasses: FreeTimeOrganizer, SeminarPlace and
Speaker. The instances of these classes contain contact and location information about the
services.

![Hierarchy of Evaluation-ontology classes.](image-url)
Figure 12. Class relations. All the classes and their relations are shown in the upper image. A more specific view about the relations of Employee and Evaluation classes is shown in the lower image.

Figure 12 and Figure 13 shows the relations (properties) between the different classes (resources). An employee evaluates the different aspects of the seminar (Evaluation-class), an employee belongs to a certain area of organisation (Organization-class) and an employee takes part in a certain seminar (Seminar-class). For making querying easier, the evaluations have been connected to several classes. (These connections could have been left out of ontology, because the information that they contain could be queried utilising other information in the ontology.) An evaluation belongs to a certain area of organisation, the seminar has an evaluation of the whole seminar, the program has
evaluations about the preinterest, content and free time of the seminar. The Speaker has evaluations of the lecturer, the SeminarPlace has evaluations about the accommodation, lecture room, food, and whole seminar place. The Speaker and SeminarPlace classes are subclasses of ServiceSupplier-class. Seminar (Seminar-class) has the certain program (Program-class). Program has a lecturer (Speaker-class) and someone who has arranged the program of free time (FreeTimeOrganizer-class).

Figure 13. Classes, their properties and relations.
5.3.2 Data Import

The evaluations were collected utilising the XML-case. The data was imported into RDF model with the help of the Protégé-2000 form interface during ontology development. An example of the Protégé-2000 form interface with some imported data is shown in Figure 14. Only a small portion of the data was imported in order to test RDF model.

![Protégé-2000 form interface](image)

*Figure 14. An example of the Protégé-2000 form interface.*

The RDF Schema (The evaluation RDF(S) file is shown in Appendix A) was created with the Protégé-2000 ontology editor. All the instances were saved into a RDF file. Both the RDF Schema and RDF data files were imported into the Sesame RDF repository. The files were imported through the Sesame web interface. Sesame was installed on top of the MySQL-database. A database called Evaluation was created into the MySQL-database. All the data were saved into the database through Sesame.
Queries of opinions were evaluated through the Sesame RQL-editor. Same queries can be made as http-queries straight from the browser.

5.3.3 Querying Possibilities of RDF Experiment

The RDF experiment gives users the possibilities to make queries into resulting knowledge base in several ways.

1. It is possible to view evaluation results based on certain lecturer, seminar location or accommodation. The query can be refined based on background information of evaluators.

A query example:

Give the evaluation of lecturer Lasse Kivikko?

Query form:

View evaluation results based on certain lecturer, seminar location or accommodation

Lecturer and topic

☑ Lasse Kivikko, Otakon Oy: Innovations

☐ Juha Wiskari, Fusan Centre of Finland: BrainPower and Intellectual Capital

☐ Esa Searinne, Teknillinen Korkeakoulu: Innostus ja luovuus tutkimustyössä

Seminar location

☐ Kesikkariono Aurantola, Jala

☐ Rymättylään Herrankukkar

☐ Kyllähötel, Rantasipi Eden, Nokia

Accommodation

☐ Cottages

☐ Cottages in village of fishermen

☐ Hotel
Query as a select-clause:

select PERSONNAME, GRADE from ev:Speaker {SPEAKER}. ev:personName {PERSONNAME}, {SPEAKER} ev:hasLecturerEvaluation {EVALUATIONS}, {EVALUATIONS} ev:grade {GRADE} where PERSONNAME="Lasse Kivikko" using namespace ev= http://tte4145.tte.vtt.fi:8080/evaluation/ontology/evaluation#

Query as an http-query:

http://tte4145.tte.vtt.fi:8080/sesame/servlets/performRqlQuery?format=html&repository=evaluation-db&query=select+PERSONNAME%2C+GRADE+from+ev%3ASpeaker+%7BSPEAKER%7D.+ev%3ApersonName+%7BPERSONNAME%7D+%2C+%7BSPEAKER%7D+ev+%3AhasLecturerEvaluation+%7BEVALUATIONS%7D+%2C+%7BEVALUATIONS%7D+ev+%3AGrade+%7BGRADE%7D+where+PERSONNAME+%3D+%22Lasse+Kivikko%22+using+namespace+ev+%3D+http%3A%2F%2Ftte4145.tte.vtt.fi:8080%2Fevaluation%20ontology%2Fevaluation%23

The answer to the query as an HTML table:

<table>
<thead>
<tr>
<th>PERSONNAME</th>
<th>GRADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lasse Kivikko&quot;</td>
<td>'3'</td>
</tr>
<tr>
<td>&quot;Lasse Kivikko&quot;</td>
<td>'4'</td>
</tr>
</tbody>
</table>

2 results found in 47 ms.

The answer to the query as an RDF-bag:

```xml
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
- <rdf:Bag rdf:ID="query_result">
  - <rdf:li>
    - <rdf:Seq>
      <rdf:li>Lasse Kivikko</rdf:li>
      <rdf:li>3</rdf:li>
    </rdf:Seq>
  </rdf:li>
  - <rdf:li>
    - <rdf:Seq>
      <rdf:li>Lasse Kivikko</rdf:li>
      <rdf:li>4</rdf:li>
    </rdf:Seq>
  </rdf:li>
</rdf:Bag>
<!> 2 results found in 16 ms  <!>
```

The query above can be refined with the evaluators' background information.

Example query: See what kinds of people have given the evaluations of Lasse Kivikko?
Query form:

Target evaluation results based on backgound information of evaluators

Background of evaluator

- Do not show any background information about evaluators
- Show all background informations about evaluators
- Select background information you are interested in

☑ Age

- All
- less than 25
- 25-30
- 31-35

☑ Gender

- all
- female
- male

☑ Education

- all
- student
- graduate engineer
- licenciate or doctor

☑ Experience at VTT

- all
- less than 1
- 1-5
- 6-10

☐ Activity of taking part to arranged TTE happenings

- all
- always
- often
- sometimes

Query:

select PERSONNAME, GRADE, AGE, EDUCATION, EXPERIENCE, GENDER
from ev:Speaker {SPEAKER}. ev:personName {PERSONNAME},
{SPEAKER} ev:hasLecturerEvaluation {EVALUATIONS}, {EVALUATIONS}
ev:grade {GRADE},
XMLDatabases XML and RDF Databases

1.0

VTT Information Technology

Modified on 10.06.03

{EVLECTURER} ev:evaluateLecturer {EVALUATIONS},{EVLECTURER} ev:age {AGE},{EVLECTURER} ev:education {EDUCATION},
{EVLECTURER} ev:experience {EXPERIENCE},{EVLECTURER} ev:gender {GENDER}
where PERSONNAME="Lasse Kivikko"
using namespace ev= http://tte4145.tte.vtt.fi:8080/evaluation/ontology/evaluation#

Answer:

<table>
<thead>
<tr>
<th>PERSONNAME</th>
<th>GRADE</th>
<th>AGE</th>
<th>EDUCATION</th>
<th>EXPERIENCE</th>
<th>GENDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lasse Kivikko&quot;</td>
<td>3</td>
<td>&quot;35-40&quot;</td>
<td>&quot;graduateEngineer&quot;</td>
<td>&quot;11-15&quot;</td>
<td>&quot;male&quot;</td>
</tr>
<tr>
<td>&quot;Lasse Kivikko&quot;</td>
<td>4</td>
<td>&quot;48-50&quot;</td>
<td>&quot;Other&quot;</td>
<td>&quot;16-25&quot;</td>
<td>&quot;male&quot;</td>
</tr>
</tbody>
</table>

2. Another possibility is to view the evaluation results based on the area and evaluation criteria (Choose an evaluation and see what kind of people gave it.) The query can be limited to certain evaluation values. It is possible to choose two evaluations and specify their values (e.g. high lecturer, low accommodation). The user gets a list of evaluations that meet the criteria.

Example query: Show the opinions about the whole TTE1 seminar?
Query form:

View evaluation results based on certain areas and evaluation criteria

Evaluation criteria (Search can be targeted with certain grade value)

- Whole seminar
  - Grade
- Preinterest of seminar program
  - Grade
- Content of seminar program
  - Grade
- Freetime Program
  - Grade
- Lecturer
  - Grade
- Food
  - Grade
- Seminar location
  - Grade
- Accommodation
  - Grade

Query:

select AREA, GRADE from ev:Employee {EMPLOYEE}. ev:belongsToArea {AREAS}, {AREAS} ev:area {AREA}, {EMPLOYEE} ev:evaluateWholeSeminar {WHOLESEMINAR}, {WHOLESEMINAR} ev:grade {GRADE}
where AREA="TTE1"
using namespace ev= http://tte4145.tte.vtt.fi:8080/evaluation/ontology/evaluation#

Answer:

<table>
<thead>
<tr>
<th>AREA</th>
<th>GRADE</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;TTE1&quot;</td>
<td>&quot;3&quot;</td>
</tr>
<tr>
<td>&quot;TTE1&quot;</td>
<td>&quot;4&quot;</td>
</tr>
</tbody>
</table>

2 results found in 47 ms.
The query can be refined with evaluators’ background information and with certain evaluation values.

*Example:* Show education and experience of people who gave the grade 3?

*Query form:*

**Target evaluation results based on background information of evaluators**

- **Background of evaluator**
  - Do not show any background information about evaluators
  - Show all background informations about evaluators
  - Select background information you are interested in

- **Age**
  - All
  - Less than 25
  - 26-30
  - 31-35

- **Gender**
  - All
  - Female
  - Male

- **Education**
  - All
  - Student
  - Graduate Engineer
  - Licentiate or Doctor

- **Experience at VTT**
  - All
  - Less than 1
  - 1-5
  - 6-10

- **Activity of taking part to arranged TTE happenings**
  - All
  - Often
  - Sometimes
View evaluation results based on certain areas and evaluation criteria

Area

Evaluation criteria (Search can be targeted with certain grade value)

- Whole seminar
- Grade

- Participant interest of seminar program
- Grade

- Content of seminar program
- Grade

- Lecture program
- Grade

- Lecturer
- Grade

- Seminar location
- Grade

- Accommodation
- Grade

Query:

select AREA, GRADE, EDUCATION, EXPERIENCE from ev:Employee
{EMPLOYEE}, ev:belongsToArea {AREAS}, {AREAS} ev:area {AREA},
{EMPLOYEE} ev:evaluateWholeSeminar {WHOLESEMINAR}, {WHOLESEMINAR}
ev:grade {GRADE}, {EMPLOYEE} ev:education {EDUCATION}, {EMPLOYEE}
ev:experience {EXPERIENCE}
where AREA="TTE1" and GRADE="3"
using namespace ev= http://tte4145.tte.vtt.fi:8080/evaluation/ontology/evaluation#
3. The third possibility is to define a group of people who have the background we are interested in, and to see what kind of evaluations they have given.

For example:

Select: Area -> Gender -> Age group -> Activity (the selection may be stopped earlier or continued further)

Choose: some Evaluation: (e.g. Accommodation)

Get: List of evaluations.

Optionally, one could continue with viewing what other evaluations this group has given.

Example query: Show opinion of those people whose age is 36-40 about the free-time program?

Query form:

Target evaluation results based on background information of evaluators

Background of evaluator:

- Do not show any background information about evaluators
- Show all background information about evaluators
- Select background information you are interested in

Query:

select AREA, GRADE, AGE from ev:Employee {EMPLOYEE}, ev:belongsToArea {AREAS}, {AREAS} ev:area {AREA}, {EMPLOYEE} ev:evaluateFreeTimeProgram {FREETIME}, {FREETIME} ev:grade {GRADE}, {EMPLOYEE} ev:age {AGE} where AGE="36-40"

using namespace ev= http://tte4145.tte.vtt.fi:8080/evaluation/ontology/evaluation#
Answer:

<table>
<thead>
<tr>
<th>AREA</th>
<th>GRADE</th>
<th>AGE</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;TTE1&quot;</td>
<td>&quot;5&quot;</td>
<td>&quot;36-40&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;TTE4&quot;</td>
<td>&quot;4&quot;</td>
<td>&quot;36-40&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;TTE1&quot;</td>
<td>&quot;4&quot;</td>
<td>&quot;36-40&quot;</td>
<td></td>
</tr>
</tbody>
</table>

3 results found in 31 ms.

Targeted query: Show also their activity to take part in TTE arranged events?

Query:

```sql
select AREA, GRADE, AGE, ACTIVITY from ev:Employee {EMPLOYEE}.
  ev:belongsToArea {AREAS}, {AREAS} ev:area {AREA}, {EMPLOYEE}
  ev:evaluateFreeTimeProgram {FREETIME}, {FREETIME} ev:grade
  {GRADE}, {EMPLOYEE} ev:age {AGE}, {EMPLOYEE} ev:activity {ACTIVITY}
  where AGE="36-40"
using namespace ev= http://tte4145.tte.vtt.fi:8080/evaluation/ontology/evaluation#
```

Answer:

<table>
<thead>
<tr>
<th>AREA</th>
<th>GRADE</th>
<th>AGE</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;TTE1&quot;</td>
<td>&quot;5&quot;</td>
<td>&quot;36-40&quot;</td>
<td>&quot;always&quot;</td>
</tr>
<tr>
<td>&quot;TTE4&quot;</td>
<td>&quot;4&quot;</td>
<td>&quot;36-40&quot;</td>
<td>&quot;always&quot;</td>
</tr>
<tr>
<td>&quot;TTE1&quot;</td>
<td>&quot;4&quot;</td>
<td>&quot;36-40&quot;</td>
<td>&quot;always&quot;</td>
</tr>
</tbody>
</table>

0 results found in 63 ms.

5.3.4 Conclusions of RDF Experiment

The RDF implementation used in this experiment gives an interesting tool to investigate evaluation results on the level user wants. The background information about the evaluators can be well utilised. This RDF-implementation was only a small experiment which would needs further work, but it proves that this kind of solutions works well and gives flexible opportunities to investigate the evaluation results. The way results has been shown can be made more sophisticated for example by including links to more detailed evaluator information, or sorting the results in different, user requested ways. It is also possible to include some statistical information to get overview of all results.

The Sesame RDF repository was tested only by using its own user interfaces. The next phase would be test RDF-model and the repository programmatically by developing Java programs to save, retrieve, manipulate and show the data.
6 Conclusions

The goal of the project was to get an assessment of the status of XML support in databases (so called mainstream databases, and XML databases), and to compare the differences in XML and RDF applications and databases. Here, also some prototype development was made.

XML support is already available for many mainstream databases, whereas RDF is only supported in some few databases that are of experimental nature. Even though RDF utilises the XML serialization syntax, the data model is very different from the XML data model. The XML data model is a tree, and that of RDF a labelled directed graph. Therefore, the RDF information can be much better utilised when the tools understand its semantic structure.

RDF can be used to manage distributed network like information better than basic XML. RDF is also gaining importance as an ontology language. With ontologies the interoperability of different applications can be improved.

The case application dealt with user opinions. This kind of information can be classified as metadata, and RDF is very well suited to expressing it. More RDF information can be added, and the data can be navigated from many directions. Unfortunately the tools to manipulate and store RDF data are in very early stage, resembling the situation with XML tools some four to five years ago. It is important to keep track of the system development in that area, and make test implementations.

Xindice was utilised as an XML database in the project. This database is very well suited to research and prototype applications, because no schema is mandatory for the data. This means that the information content can be varied and modified during the development and pilot phase. This adds flexibility compared to utilising a database that requires a schema or a relational database.
References


Appendix A: Evaluation RDF Schema File

<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE rdf:RDF [ 
<!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
<!ENTITY a 'http://protege.stanford.edu/system#'>
<!ENTITY evaluation 'http://tte4145.tte.vtt.fi:8080/evaluation/ontology/evaluation#'>
<!ENTITY rdfs 'http://www.w3.org/TR/1999/PR-rdf-schema-19990303#'> ]>
<rdf:RDF xmlns:rdf="&rdf;"
         xmlns:a="&a;"
         xmlns:evaluation="&evaluation;"
         xmlns:rdfs="&rdfs;">
  <rdfs:Class rdf:about="&evaluation;Accommodation"
             rdfs:label="Accommodation">
    <rdfs:subClassOf rdf:resource="&evaluation;Evaluation"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&evaluation;Content"
             rdfs:label="Content">
    <rdfs:subClassOf rdf:resource="&evaluation;Evaluation"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&evaluation;Employee"
             rdfs:label="Employee">
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&evaluation;Evaluation"
             rdfs:label="Evaluation">
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&evaluation;Food"
             rdfs:label="Food">
    <rdfs:subClassOf rdf:resource="&evaluation;Evaluation"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&evaluation;FreeTimeOrganizer"
             rdfs:label="FreeTimeOrganizer">
    <rdfs:subClassOf rdf:resource="&evaluation;ServiceSupplier"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&evaluation;FreeTimeProgram"
             rdfs:label="FreeTimeProgram">
    <rdfs:subClassOf rdf:resource="&evaluation;Evaluation"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&evaluation;LectureRoom"
             rdfs:label="LectureRoom">
    <rdfs:subClassOf rdf:resource="&evaluation;Evaluation"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&evaluation;Lecturer"
             rdfs:label="Lecturer">
    <rdfs:subClassOf rdf:resource="&evaluation;Evaluation"/>
  </rdfs:Class>
  <rdfs:Class rdf:about="&evaluation;Organization"
<rdf:Property rdf:about="&evaluation;age"
    a:maxCardinality="1"
    rdfs:label="age">
     <rdfs:domain rdf:resource="&evaluation;Employee"/>
     <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&evaluation;area"
    a:maxCardinality="1"
    rdfs:label="area">
     <rdfs:domain rdf:resource="&evaluation;Organization"/>
     <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&evaluation;belongsToArea"
    a:maxCardinality="1"
    rdfs:label="belongsToArea">
     <rdfs:domain rdf:resource="&evaluation;Employee"/>
     <rdfs:range rdf:resource="&evaluation;Organization"/>
     <rdfs:domain rdf:resource="&evaluation;Seminar"/>
</rdf:Property>

<rdf:Property rdf:about="&evaluation;comment"
    a:maxCardinality="1"
    rdfs:label="comment">
     <rdfs:domain rdf:resource="&evaluation;Evaluation"/>
     <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&evaluation;companyName"
    a:maxCardinality="1"
    rdfs:label="companyName">
     <rdfs:domain rdf:resource="&evaluation;ServiceSupplier"/>
     <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&evaluation;date"
    a:maxCardinality="1"
    rdfs:label="date">
     <rdfs:domain rdf:resource="&evaluation;Seminar"/>
     <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&evaluation;description"
    a:maxCardinality="1"
    rdfs:label="description">
     <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&evaluation;education"
    a:maxCardinality="1"
    rdfs:label="education">
     <rdfs:domain rdf:resource="&evaluation;Employee"/>
     <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&evaluation;email"
    a:maxCardinality="1"
    rdfs:label="email">
     <rdfs:domain rdf:resource="&evaluation;ServiceSupplier"/>
     <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

<rdf:Property rdf:about="&evaluation;evaluateAccommodation"
    a:maxCardinality="1"
    rdfs:label="evaluateAccommodation">
     <rdfs:range rdf:resource="&evaluation;Accommodation"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;evaluateContent"
a:maxCardinality="1"
rdfs:label="evaluateContent">
  <rdfs:range rdf:resource="&evaluation;Content"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;evaluateFood"
a:maxCardinality="1"
rdfs:label="evaluateFood">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&evaluation;Food"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;evaluateFreeTimeProgram"
a:maxCardinality="1"
rdfs:label="evaluateFreeTimeProgram">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&evaluation;FreeTimeProgram"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;evaluateLecturer"
a:maxCardinality="1"
rdfs:label="evaluateLecturer">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&evaluation;Lecturer"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;evaluateLecturerRoom"
a:maxCardinality="1"
rdfs:label="evaluateLecturerRoom">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&evaluation;LectureRoom"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;evaluateOther"
a:maxCardinality="1"
rdfs:label="evaluateOther">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&evaluation;Other"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;evaluatePlace"
a:maxCardinality="1"
rdfs:label="evaluatePlace">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&evaluation;Place"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;evaluatePreInterest"
a:maxCardinality="1"
rdfs:label="evaluatePreInterest">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&evaluation;PreInterest"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;evaluateWholeSeminar"
a:maxCardinality="1"
rdfs:label="evaluateWholeSeminar">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&evaluation;WholeSeminar"/>
</rdf:Property>
<rdfs:domain rdf:resource="&evaluation;Employee"/>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;experience"
a:maxCardinality="1"
rdfs:label="experience">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;freeTimeDescription" a:maxCardinality="1" rdfs:label="freeTimeDescription">
  <rdfs:domain rdf:resource="&evaluation;Program"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;gender" a:maxCardinality="1" rdfs:label="gender">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;grade" a:maxCardinality="1" a:range="integer" rdfs:label="grade">
  <rdfs:domain rdf:resource="&evaluation;Evaluation"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;hasAccommodationEvaluation" rdfs:label="hasAccommodationEvaluation">
  <rdfs:domain rdf:resource="&evaluation;Accommodation"/>
  <rdfs:range rdf:resource="&evaluation;SeminarPlace"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;hasContentEvaluation" rdfs:label="hasContentEvaluation">
  <rdfs:domain rdf:resource="&evaluation;Content"/>
  <rdfs:range rdf:resource="&evaluation;Program"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;hasEvaluation" a:maxCardinality="1" rdfs:label="hasEvaluation">
  <rdfs:domain rdf:resource="&evaluation;Evaluation"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;hasFoodEvaluation" rdfs:label="hasFoodEvaluation">
  <rdfs:domain rdf:resource="&evaluation;Food"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;hasFreeTimeEvaluation" rdfs:label="hasFreeTimeEvaluation">
  <rdfs:domain rdf:resource="&evaluation;FreeTimeProgram"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;hasFreeTimeOrganizer" a:maxCardinality="1" rdfs:label="hasFreeTimeOrganizer">
  <rdfs:domain rdf:resource="&evaluation;FreeTimeOrganizer"/>
</rdf:Property>
</rdf:Property>
<rdf:Property rdf:about="&evaluation;hasLecturer" a:maxCardinality="1" rdfs:label="hasLecturer">
  <rdfs:domain rdf:resource="&evaluation;Program"/>
</rdf:Property>
<rdfs:range rdf:resource="&evaluation;Speaker"/>
</rdf:Property>

<rdfs:range rdf:resource="&evaluation;Lecturer"/>
</rdf:Property>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;Speaker"/>
</rdf:Property>

<rdfs:range rdf:resource="&evaluation;LectureRoom"/>
</rdf:Property>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;SeminarPlace"/>
</rdf:Property>

<rdfs:range rdf:resource="&evaluation;Place"/>
</rdf:Property>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;SeminarPlace"/>
</rdf:Property>

<rdfs:range rdf:resource="&evaluation;PreInterest"/>
</rdf:Property>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;Program"/>
</rdf:Property>

<rdfs:range rdf:resource="&evaluation;Program"/>
</rdf:Property>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;Seminar"/>
</rdf:Property>

<rdfs:range rdf:resource="&evaluation;Place"/>
</rdf:Property>
</rdf:Property>

<rdfs:range rdf:resource="&evaluation;LecturerRoomEvaluation"/>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;Speaker"/>
</rdf:Property>

<rdfs:range rdf:resource="&evaluation;SeminarPlace"/>
</rdf:Property>
</rdf:Property>

<rdfs:range rdf:resource="&evaluation;WholeSeminarEvaluation"/>
</rdf:Property>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;Evaluation"/>
</rdf:Property>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;SeminarPlace"/>
</rdf:Property>

<rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;Program"/>
</rdf:Property>
</rdf:Property>

<rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
</rdf:Property>

<rdfs:domain rdf:resource="&evaluation;PersonName"/>
a:maxCardinality="1"
rdfs:label="personName">
  <rdfs:domain rdf:resource="&evaluation;ServiceSupplier"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;placeDescription"
a:maxCardinality="1"
rdfs:label="placeDescription">
  <rdfs:domain rdf:resource="&evaluation;SeminarPlace"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;placeName"
a:maxCardinality="1"
rdfs:label="placeName">
  <rdfs:domain rdf:resource="&evaluation;ServiceSupplier"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;preInterest"
a:maxCardinality="1"
rdfs:label="preInterest">
  <rdfs:range rdf:resource="&evaluation;PreInterest"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;restaurantDescription"
a:maxCardinality="1"
rdfs:label="restaurantDescription">
  <rdfs:domain rdf:resource="&evaluation;SeminarPlace"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;seminarTopic"
a:maxCardinality="1"
rdfs:label="seminarTopic">
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;takePart"
a:maxCardinality="1"
rdfs:label="takePart">
  <rdfs:domain rdf:resource="&evaluation;Employee"/>
  <rdfs:range rdf:resource="&evaluation;Seminar"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;telephone"
a:maxCardinality="1"
rdfs:label="telephone">
  <rdfs:domain rdf:resource="&evaluation;ServiceSupplier"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;topicDescription"
a:maxCardinality="1"
rdfs:label="topicDescription">
  <rdfs:domain rdf:resource="&evaluation;Program"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;topicName"
a:maxCardinality="1"
rdfs:label="topicName">
  <rdfs:domain rdf:resource="&evaluation;Program"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>

</rdf:Property>
<rdf:Property rdf:about="&evaluation;webPage"
<rdf:RDF>
  <rdf:Property a:maxCardinality="1" rdfs:label="webPage">
    <rdfs:domain rdf:resource="&evaluation;ServiceSupplier"/>
    <rdfs:range rdf:resource="&rdfs;Literal"/>
  </rdf:Property>

  <rdf:Property rdf:about="&evaluation;wholeSeminar" a:maxCardinality="1" rdfs:label="wholeSeminar">
    <rdfs:range rdf:resource="&evaluation;WholeSeminar"/>
  </rdf:Property>
</rdf:RDF>
Appendix B: XML Schema File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Tarja Hänninen (VTT Information Technology) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="Evaluation">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Area"/>
        <xs:element ref="WholeSeminar"/>
        <xs:element ref="PreInterest"/>
        <xs:element ref="Content"/>
        <xs:element ref="FreeTimeProgram"/>
        <xs:element ref="Lecturer"/>
        <xs:element ref="Food"/>
        <xs:element ref="Accomodation"/>
        <xs:element ref="LectureRoom"/>
        <xs:element ref="Place"/>
        <xs:element ref="Other" minOccurs="0"/>
        <xs:element ref="Background"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Area">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="TTE1"/>
        <xs:enumeration value="TTE2"/>
        <xs:enumeration value="TTE3"/>
        <xs:enumeration value="TTE4"/>
        <xs:enumeration value="TTE5"/>
        <xs:enumeration value="TTE6"/>
        <xs:enumeration value="TTE7"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>
  <xs:element name="WholeSeminar">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Grade" minOccurs="0"/>
        <xs:element ref="Comment" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="PreInterest">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Grade" minOccurs="0"/>
        <xs:element ref="Comment" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<xs:element name="Other" type="xs:string"/>
<xs:element name="Grade">
    <xs:simpleType>
        <xs:restriction base="xs:int">
            <xs:enumeration value="1"/>
            <xs:enumeration value="2"/>
            <xs:enumeration value="3"/>
            <xs:enumeration value="4"/>
            <xs:enumeration value="5"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="Comment" type="xs:string"/>
<xs:element name="Background">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Age"/>
            <xs:element ref="Gender"/>
            <xs:element ref="Experience"/>
            <xs:element ref="Education"/>
            <xs:element ref="Activity"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Age">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="less than 25"/>
            <xs:enumeration value="26-30"/>
            <xs:enumeration value="31-35"/>
            <xs:enumeration value="36-40"/>
            <xs:enumeration value="41-45"/>
            <xs:enumeration value="46-50"/>
            <xs:enumeration value="51-55"/>
            <xs:enumeration value="more than 55"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="Gender">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="male"/>
            <xs:enumeration value="female"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="Experience">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="less than 1"/>
            <xs:enumeration value="1-5"/>
            <xs:enumeration value="6-10"/>
            <xs:enumeration value="11-15"/>
            <xs:enumeration value="16-25"/>
            <xs:enumeration value="more than 26"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="Education">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="student"/>
      <xs:enumeration value="graduate engineer"/>
      <xs:enumeration value="licenciate or doctor"/>
      <xs:enumeration value="other"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="Activity">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="always"/>
      <xs:enumeration value="often"/>
      <xs:enumeration value="sometimes"/>
      <xs:enumeration value="seldom"/>
      <xs:enumeration value="never"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
</xs:schema>