

# NESSTAR: A Semantic Web Application for Statistical Data and Metadata

*Pasqualino 'Titto' Assini (titto@nesstar.com)*

*Nesstar Ltd*

## 1 Statistical Data Dissemination: The Reality and the Dream

The social sciences are big producers and consumers of statistical data. Surveys, censuses and opinion polls are the basic sources for most quantitative research in sociology and economy. These data are collected and preserved by specialised Data Archives and traditionally disseminated to researchers as datafiles stored on some form of magnetic media and accompanied by bulky printed documentation. Given that there are many such Archives, each one with its own distinct access and dissemination procedures, it is often not easy for a researcher to find and get hold of the right information.

In 1998 the European Union funded a research and development project named Networked Social Science Tools and Resources (NESSTAR). The aim of the project was to bring the advantages of "instant access" to the world of statistical data dissemination. At the time the WWW had already made the publishing of textual and graphical information easier and cheaper than ever. Huge amount of information had been made available world-wide at a press of a button and at virtually no cost. The question that NESSTAR was called to answer was if it was possible to create a "Data Web" that would make just as easy to publish, locate and access statistical data.

NESSTAR provides at least some of the basic element of the Data Web "dream" and does so by building on both the current WWW infrastructure and the new Semantic Web technologies.

The NESSTAR project has been followed by FASTER [2], another European project that has further developed the Data Web concept and implementation. NESSTAR and FASTER have been sufficiently successful to convince two of their main contractors, the University of Essex in England and the University of Bergen in Norway, to spin off a company to exploit commercially the Data Web technology. Nesstar Ltd is currently busy developing Data Web solutions for a number of international clients.

## 2 The Conceptual Model

The conceptual core of NESSTAR is an Object Model of statistical data and metadata<sup>1</sup>. Though not very extensive, it currently contains about 15 classes, it captures many of the key domain concepts: statistical studies, datafiles, statistical variables, indicators, tables, etc. The classes are linked together by a set of relationships: a Study for example contains Cubes each having one or more Dimensions, etc. Some objects have also methods. Statistical Studies for example have methods such as Tabulate or Frequency and other common statistical operations.

In addition to the domain specific concepts the model has another 10 or so domain independent "support" classes. An example is the Server class. It represents the server where the objects are hosted and provides basic administrative functions such as file transfer, server reboot and shutdown, etc. It also plays, in the Data Web, a role similar to that of the home page in the normal Web. From a home page you can, by following hyperlinks, reach all the contents of a web site. Similarly, starting from a server object an application can, by recursively traversing the object relationships, reach all the other objects hosted by the server. Another generic class is Catalog. Catalogs are used to group objects, just like folders in a filesystem. Catalogs can be browsed, an application can get the complete list of all the objects contained in a catalog, or searched to select only the objects that satisfy a particular condition.

Many statistical studies contain sensitive information that cannot be made available without restrictions. For this reason the model includes a set of objects to represent Users, the Roles they play in the system (example: Administrator, FinalUser, DataPublisher), the agreements that they have accepted (such as: "I agree to use these data only for non commercial research purposes"), etc. On top of this small security model is possible to define a variety of access control policies<sup>2</sup>.

## 3 Making It Real: Putting Semantic Web Technology to Work

The *modus operandi* of NESSTAR is very simple. Data publishers make their statistical information available as objects, as specified by the Nesstar Object Model, on the Net. Each publisher runs its own server. Users use the system pretty much as they use the Web: if they know where some information is stored they can "point" their client application to it (for example by typing the object URL in a location bar or by clicking on a hyperlink). The client will access the remote statistical object and display it to the user. The users can also perform searches to find object with particular characteristics

---

<sup>1</sup> For reasons of space the object model class diagram is not included in this paper. It can be found at [http://www.nesstar.org/sdk/nesstar\\_object\\_model.pdf](http://www.nesstar.org/sdk/nesstar_object_model.pdf)

<sup>2</sup> NESSTAR servers support a wide range of Access Control mechanisms.

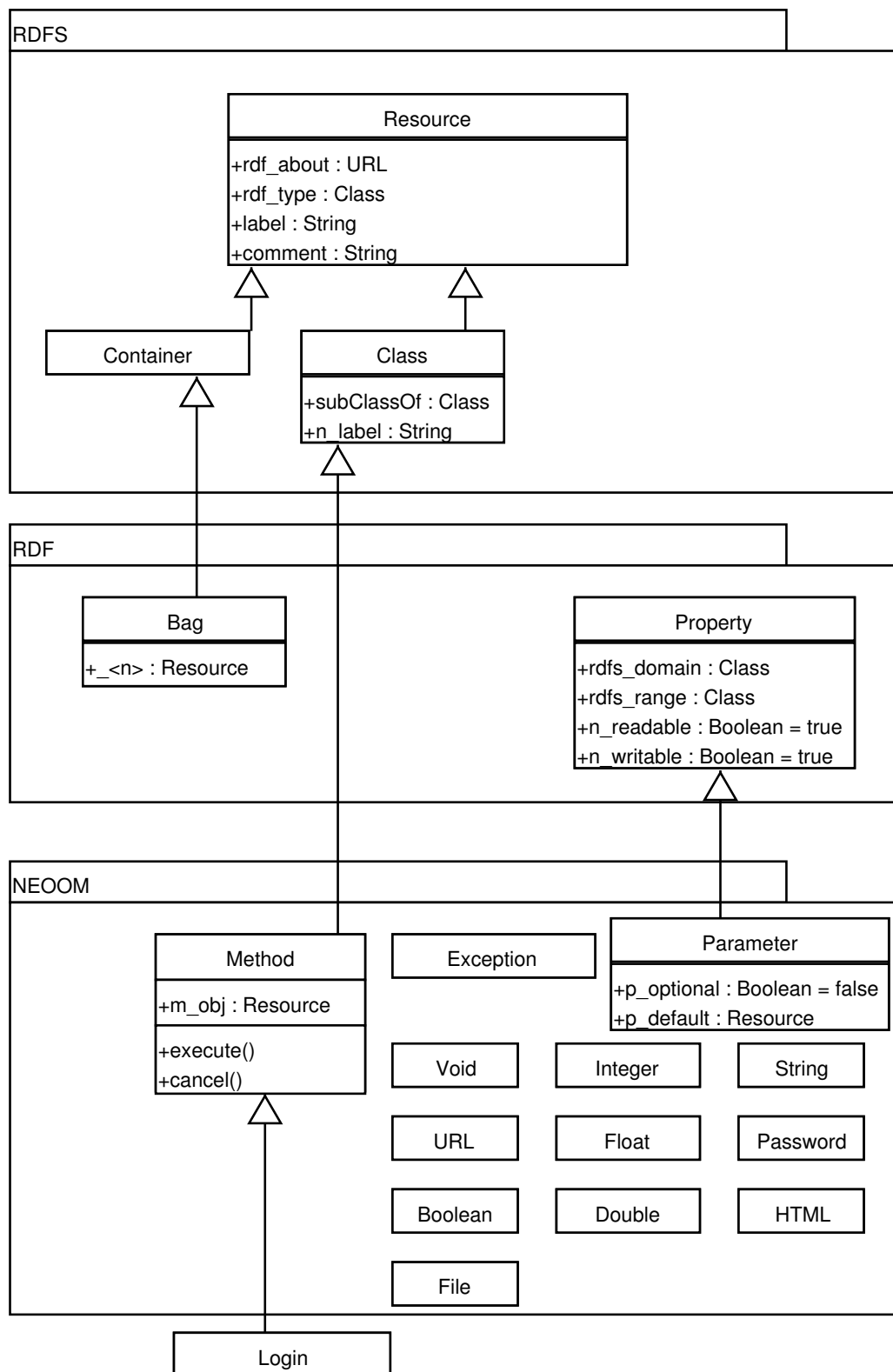


Fig. 1: NEOOM Object Model

such as: "find all variables about political orientation". This is similar to using a search engine such as Google to find all HTML pages that contain a given keyword.

The NESSTAR system is built on top of a lightweight Web and object-oriented middleware that we call NESstar Object Oriented Middleware (NEOOM). NEOOM is closely based on Web and Semantic Web standards, in particular RDF, RDF Schema, HTML and HTTP. So closely, actually, that more than a distinct framework it can be considered as just a set of guidelines on how to use off the shelf (semantic) web technology to build distributed object-oriented systems.

NEOOM is described briefly in [7] and more extensively in [6]. For the purpose of this paper I will briefly explain how NEOOM solves three basic problems:

- describing the state of objects instances
- providing a formal definition of a class properties and methods
- performing remote method calls

For the first problem the W3C provide a standard solution in the form of the RDF [11] language. RDF has two features that make it ideally suited as a representation language for our model: it is object oriented and has a standard XML representation. NESSTAR objects, just like any other Web resource, "live" at a given address (an URL). When a client access the object URL (for example using HTTP) the object returns a description of its current state in RDF.

Using RDF Schema [9] we can formally define the properties of a given class of objects and therefore represent most of the information in the NESSTAR Object Model. There is only one major omission: RDF Schema does not provide a way of describing the behaviour of an object, that is to say the operations that it can perform. NESSTAR objects can perform a wide range of operations: queries, statistical operations, file tranfers, etc. and we need a way of specifying them formally. In order to do so we have defined the NEOOM Object Model, a small 'RDF ontology' to describe methods (see [6] for details). With this extension RDF becomes a fully-fledged Interface Definition Language (IDL).

In the NEOOM Object Model (see Fig.1), methods (e.g. *Login*) are defined as subclasses of the *Method* class. Method invocations are represented by instances of the method classes. Defining a method as a class is a bit unusual<sup>3</sup> but it has the advantage of making a method invocation an object in its own right. Being an object a method invocation can be represented

---

<sup>3</sup>In Java for example, a method is represented by an instance of `nesstar.lang.reflect.Method`, not by a separate class.

in RDF (and therefore stored, transmitted or logged easily) and can have methods and properties.

Method parameters are defined by instances of the *Parameter* class. A *Parameter* is conceptually very similar to an *rdf:property* and it inherits from it. As methods are classes then a parameter is just one of its instance properties.

The last problem consists in performing remote object-oriented calls. SOAP [5] is an increasingly popular solution to this problem. Though we plan to support SOAP in the future we currently rely on an older and much simpler alternative: the protocol used to submit HTML Forms (as specified in [12, sect17]). It's easy to define a set of conventions to map method calls on top of this simple protocol (again see [6] for details). This solution has a couple of significant advantages with respect to sending a SOAP XML message: method calls can be performed using a normal web browser (and in general using programs written in any language that comes with an HTTP library) and it is easy to determine an URL that corresponds to the operation. The URL can be used by an application to represent and replay the operation.

## 4 System Components

The basic architecture of the Nesstar system is identical to the WWW architecture. The information is hosted in Nesstar Servers. The Servers serve both normal WWW resources such as HTML pages, images, etc. and statistical objects. Just as the WWW, NESSTAR is at the same time fully distributed, each server is totally independent and there is no single point of failure, and integrated, as users experience it as an interconnected whole.

Users can access the system using the NESSTAR Explorer (a Java application for power users), the NESSTAR Light Explorer (a WWW interface based on Servlet technology), the NESSTAR Publisher (a metadata editing, validating and publishing tool) or the Object Browser (a web based generic client used for object testing and administration).<sup>4</sup>

The Nesstar Explorer is particularly interesting for its similarity to a common Web Browser. Users can enter the URL of any WWW or Nesstar resource in a Location bar. Normal WWW resources will be displayed just as they would in a normal Web Browser (or an external Web Browser is invoked to handle them), NESSTAR statistical objects are handled specially and displayed and manipulated in an efficient and flexible user interface.

The current version of NESSTAR is mainly implemented in Java (some modules are in C++). The deployment diagram 2 shows the main components of the system. On the server side we have three main components: an

---

<sup>4</sup> The clients can be downloaded from the Nesstar Web Site [3]. Sample Web clients (the Object Browser and the NESSTAR Light) can be accessed at <http://nesstar.data-archive.ac.uk/>

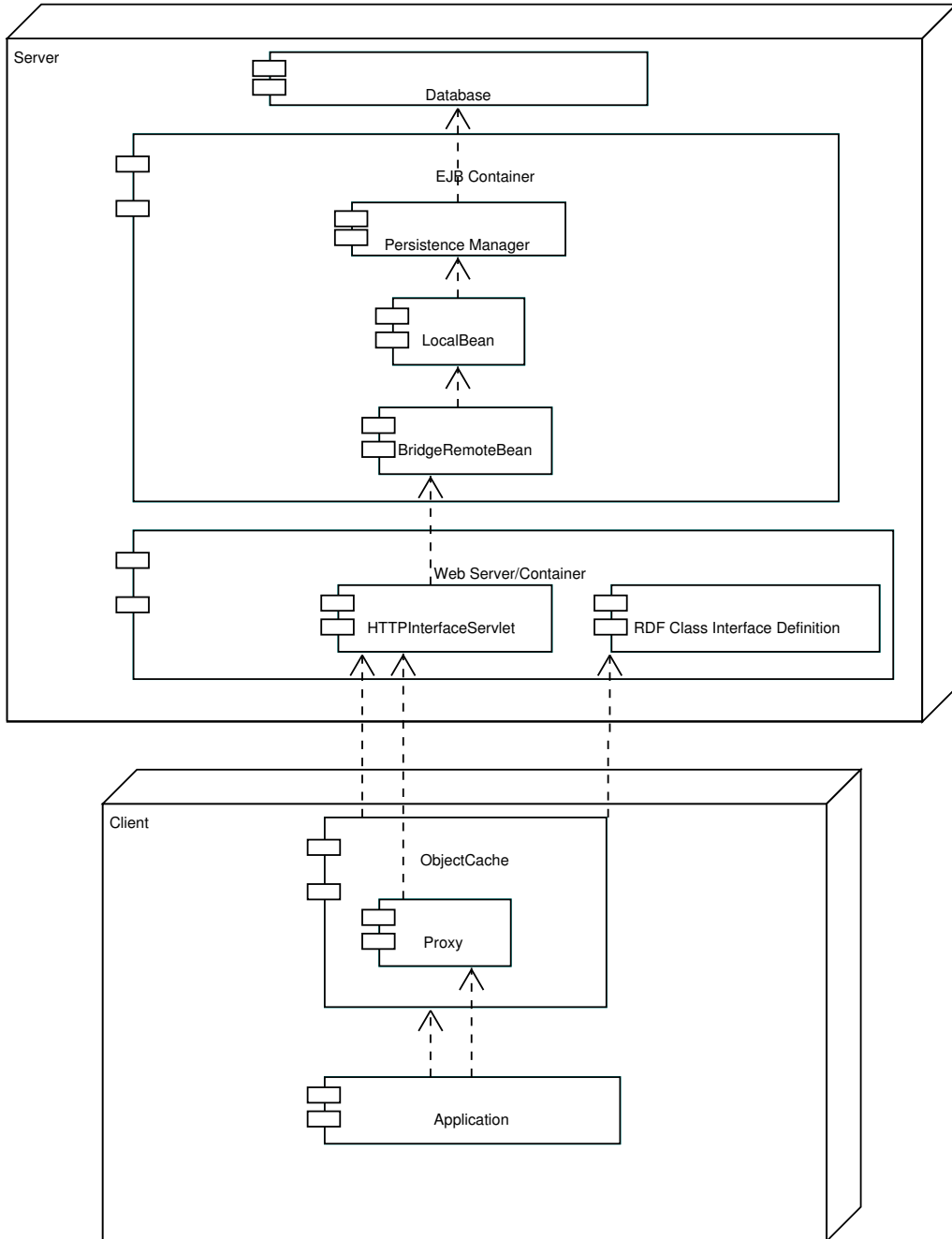


Fig. 2: System Components

Enterprise Java Bean Container [1] (JBoss 2.4.x in the current implementation), a Web Server/Container (currently Tomcat 3.x) and a relational database. The NESSTAR objects are hosted in the EJB Container as Beans. They have only local interfaces so they are not directly accessible from the outside of the container. Access takes place through a Bridge Bean. The interface with the Web is implemented by a Servlet that converts HTTP requests in RMI calls to the Bridge Bean. The RDF class interface definitions are stored in the Web Container. They are downloaded on demand by clients, such as the Object Browser, that need to discover at run-time the interface of an object.

Java client side applications access the remote NESSTAR objects through an object oriented API that supports:

- execution of remote object methods
- access of remote object properties
- traversing of object relationships
- operation bookmarks
- caching of remote objects
- handling of (multiple) authentication challenges
- HTTP and HTTPS wire protocols

For every class of NESSTAR objects the API includes a corresponding 'proxy' class. All accesses to remote objects takes place through the proxy classes. The proxies instances are hold in an object cache. When a client applications asks for an object with a given URL the API checks if the object is already in the cache. If this is not the case it performs an HTTP GET operation to the object URL. If an RDF description is found at the URL the corresponding proxy instance is automatically created, cached and returned to the calling application.

Once an application has got the proxy corresponding to the desired object it can access its properties via normal accessor methods (get/set) and perform operations on it. For each operation the API provides a corresponding URL. The application can store the operation URL so that the operation can be replayed at a later time.

## 5 Design and Development in NESSTAR

The design and development of NESSTAR objects is rather straightforward (see the use case diagram Fig.3). A Designer uses an UML modelling tool (currently we use Argo/UML or Poseidon) to create a Class Diagram specifying the classes to be created. The class diagram is saved as an XMI [4]

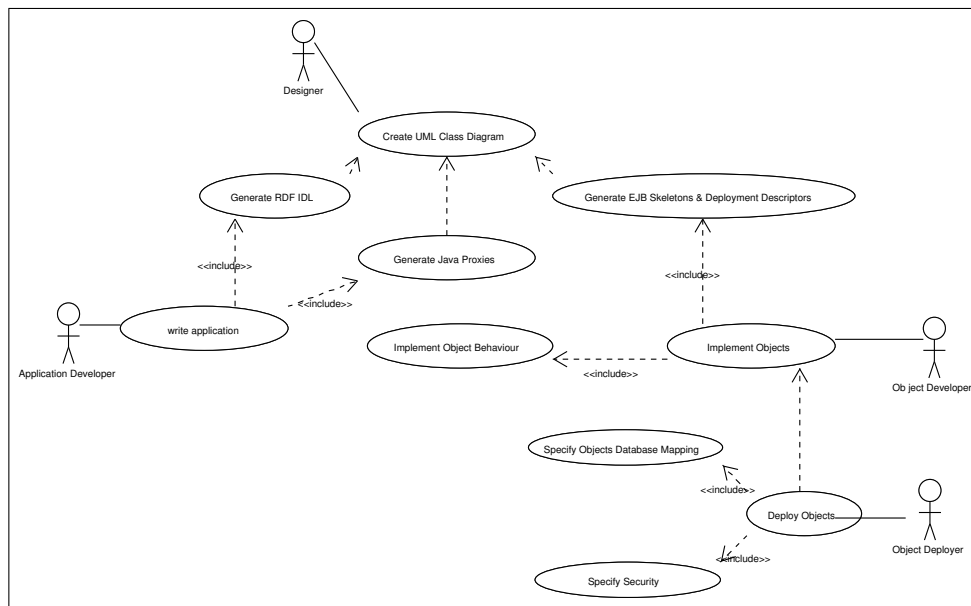


Fig. 3: Design and Development Use Cases

file, the standard format for storage of UML diagrams. The XMI file is processed, using XSLT scripts, to generate for each class:

- the Interface definition in RDF
- the Java proxy for the API
- the Skeletons of the Enterprise Java Bean plus the corresponding EJB deployment descriptor

The Object Developer can now proceed to implement the object by adding the methods implementations to the object skeletons. A Deployer completes the process and deploys the new classes in the EJB server by specifying their database mapping and security information.

The Application Developer adds the proxy classes for the objects he is interested in processing to its classpath and use them to write his application. Applications can ask the API to read in the RDF Interface Definitions to discover dynamically the properties and operations of any NESSTAR object.

## 6 Conclusions

Compared with other Semantic Web applications NESSTAR is extremely simple. It doesn't define sophisticated ontologies or make use of advanced

features of RDF such as reification. It doesn't use logical inference either. But it still satisfies the requirements of a Semantic Web application:

*The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. [10].*

All that NESSTAR aims to do is to make available, on the existing Web, a great quantity of statistical data and metadata that is currently locked in incompatible or human understandable only formats. It's a modest objective but if it were to be achieved it would revolutionize the way people access this kind of information.

I suspect that one of the reasons why the Semantic Web doesn't yet seem to be taking off while related, but much simpler technologies, such as Web Services are all the rage, is its perceived complexity. Too many people, I think, have a look at RDF or worse DAML+OIL and run away screaming. There is a real risk that the Semantic Web will be seen as a form of distributed Artificial Intelligence suitable only for the most esoteric applications or as a fancy subject for MIT PhD dissertations.

Simplicity is probably nowhere as important as in the Internet environment. A good example is given by the WWW itself, an hypertext system that provides only the most basic form of hypertext linkage: the, often broken, uni-directional link. At the beginning of the Nineties, when the WWW was born, there were a number of much more sophisticated hypertext systems available but none of them has had an impact even remotely comparable to that of the "humble" WWW. We now live in a world that has been thoroughly changed by the effects of the marriage of the Internet with such a simple technology.

Nowadays Web technology has become very complex. Gone are the days when you could learn all you needed to know about HTML and HTTP in an afternoon. But it had started small and simple. The Semantic Web has an even greater potential to change our lives for the best but it probably needs to go through the same process: start with simple, easy to understand applications and move from there.

The other key element of a successful distributed system is extensibility. A key advantage of RDF is that, using inheritance, you can extend and specialize an existing concept without breaking all the applications that depends on it. We have had a good demonstration of the power of this mechanism recently when after having massively extended the NESSTAR object model our old clients have (mostly) kept on working treating the new more specialised objects as the more general concepts they were originally developed to process.

This is one major advantage of RDF with respect to WSDL [8] as an IDL. WSDL not being object-oriented, basically is just an RPC specification

language, does not support the same kind of interface extensibility. This will create a fair amount of grief when it will come to upgrade and evolve Web Services applications.

In conclusion I think that the NESSTAR experience proves that simple Semantic Web technology can be used successfully and add significant value to real world applications.

## References

- [1] Enterprise JavaBeans. <http://java.sun.com/products/ejb/>.
- [2] EU Project FASTER. <http://www.faster-data.org>.
- [3] EU Project NESSTAR. <http://www.nesstar.org>.
- [4] XML Metadata Interchange (XMI). <http://www.oasis-open.org/cover/xmi.html>.
- [5] *Simple Object Access Protocol (SOAP) 1.1*. World Wide Web Consortium, 2000.
- [6] Pasqualino Assini. NEOOM: A Web and Object Oriented Middleware System. <http://www.nesstar.org/sdk/neoom.pdf>, 2001.
- [7] Pasqualino Assini. Objectifying the Web the 'light' way: an RDF-based framework for the description of Web objects. In *WWW10 Poster Proceedings*, Hong Kong, May 2001. World Wide Web Consortium.
- [8] Erikother Christensen, editor. *Web Services Description Language (WSDL) 1.1*. World Wide Web Consortium, March 2001.
- [9] R.V. Guha and Dan Brickley, editors. *Resource Description Framework (RDF) Schema Specification 1.0*. World Wide Web Consortium, March 2000.
- [10] James Hendler, Tim Berners-Lee, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.
- [11] Ora Lassila and Ralph R. Swick, editors. *Resource Description Framework (RDF) Model and Syntax Specification*. World Wide Web Consortium, 1999.
- [12] Dave Raggett et al., editors. *HTML 4.01 Specification*. World Wide Web Consortium, December 1999.