



Methodology and tools for exploiting the RDF schema

Distribution: Public

MedIEQ

Quality Labeling of Medical Web content using Multilingual
Information Extraction

National Centre for Scientific Research "Demokritos"
Teknillinen Korkeakoulu – Helsinki University of Technology
Universidad Nacional de Educacion a Distancia
Col.legi Oficial de Metges de Barcelona
Zentralstelle der deutschen Ärzteschaft zur Qualitätssicherung in der Medizin
Vysoka Skola Ekonomicka V Praze
I-Sieve Technologies Ltd

2005107 **D5**

February 2007

Project ref. no.	2005107
Project acronym	MedIEQ
Project full title	<i>Quality Labelling of Medical Web content using Multilingual Information Extraction</i>

Security (distribution level)	<i>Public</i>
Contractual date of delivery	<i>31 December 2006</i>
Actual date of delivery	<i>16 February 2007</i>
Deliverable number	<i>D5</i>
Deliverable name	<i>Methodology and tools for exploiting the RDF schema</i>
Type	<i>Report</i>
Status & version	<i>Final</i>
Number of pages	<i>24</i>
WP contributing to the deliverable	<i>WP4</i>
WP / Task responsible	<i>WMA</i>
Other contributors	<i>AQUMED, NCSR</i>
Author(s)	<i>Pantelis Nasikas, Vangelis Karkaletsis (NCSR), Miguel Angel Mayer (WMA), Dagmar Villarroel Gonzales (AQUMED)</i>
EC Project Officer	<i>Artur Furtado</i>
Keywords	<i>Content labels, RDF labels, label management, label generation, label validation</i>
Abstract (for dissemination)	<i>Having specified the 1st version of the MedIEQ RDF labelling schema allowing the generation and validation of RDF quality labels, we developed a toolkit, LAM (Label Management toolkit) for the management of such labels. LAM is part of the AQUA system, a system assisting the labelling of health-related web resources, by providing tools that help the identification of unlabelled web resources, automate a considerable part of the labelling process and facilitate the monitoring of already labelled resources.</i>

Table of Contents

Executive Summary	4
1. Introduction	5
2. Existing labelling mechanisms	6
3. Design Requirements for the Labels Management Toolkit.....	9
3.1. Objectives.....	9
3.2. LAM in the AQUA Process	9
3.3. User Roles and Responsibilities.....	9
3.4. Functional Requirements	9
3.5. Non Functional Requirements	10
4. Labels Management Toolkit modules	11
4.1. Label Generation.....	11
4.2. Label Uploading.....	12
4.3. Label Validation.....	13
4.4. Label Editing.....	14
4.5. Label Versioning.....	14
4.5.1 The versioning cycle	14
4.5.2 Versioning Actions on the User Interface	15
4.6. Module Administration	15
5. Labels Management Toolkit: Development framework.....	17
5.1. LAM design in general	17
5.2. Configuration	17
5.3. Database	17
5.4. EJB3 Session bean Actions, Entities and Services	18
5.5. JSF Interface	19
5.6. Libs.....	19
6. Concluding remarks.....	20
References.....	21
Appendix I – RDF-Content Labels model	22
Label Metadata.....	22
Label Restrictions/Grouping	22
Label Properties.....	22
Appendix II – MedIEQ RDF Schema.....	23

Executive Summary

Based upon state-of-the-art technology in the areas of web crawling and spidering, multilingual information extraction, semantic resources and quality labelling, MedIEQ aims to pave the way towards the automation of quality labelling process in health related web sites.

Towards this objective, MedIEQ develops tools for the creation and management of machine readable labels, using the first version of the labelling schema introduced in Deliverable D4.1 “1st version of RDF schema for medical content labels”. This schema exploits the RDF-CL [11] model, issued by the EC-funded project QUATRO [12].

The label management toolkit (LAM) is part of the integrated MedIEQ system AQUA (Assisting Quality Assessment). AQUA is designed to support the work of the labelling expert by providing tools that help the identification of unlabelled web resources, automate a considerable part of the labelling process and facilitate the monitoring of already labelled resources.

This document describes the methodology used for exploiting the labelling schema of MedIEQ focusing on the presentation of the 1st version of the label management toolkit (LAM) that implements this methodology. The design of LAM is based on the requirements of the labeling authorities participating in MedIEQ, WMA and AQuMed.

1. Introduction

In Deliverable D4.1 “1st version of RDF schema for medical content labels”, we described the main approaches in Web content characterization as well as the existing mechanisms in quality labelling of health-related Web content. We then presented the 1st version of the MedIEQ RDF vocabulary, which is based upon the work of previous projects and the re-use of existing RDF vocabularies (see Appendix I for the RDF-CL model and Appendix II for the 1st version of the MedIEQ vocabulary).

In this deliverable, we present the methodology and tools for exploiting the MedIEQ RDF schema. Section 2 outlines the existing labelling mechanisms, presents the labelling process currently used by WMA and AQuMed, and the way this process can be assisted by MedIEQ tools. In section 3, the design requirements for the labels management toolkit (LAM) that will exploit the MedIEQ schema, are introduced. Section 4 analyzes LAM modules while section 5 provides the developers manual. Finally, in section 6 we conclude by pointing out the novel aspects of our approach and the next steps.

2. Existing labelling mechanisms

There are three major mechanisms in medical quality labelling. The first one is based on third party rating where the web site is assessed by a labelling agency, in terms of certain labelling criteria such as the ones specified within the eEurope 2002 initiative, and is asked to make some changes to get the accreditation label which is then added to the web site. The second one examines medical web sites in specific thematic areas, characterizes them against certain criteria, filters some of them based on their characterization, and organizes the rest into web directories to facilitate access by health information consumers. The third mechanism is based on self-adherence to some codes of conduct or ethics that is nothing more than a claim or a pledge with little enforceability. Table 1 represents the most important accreditation initiatives and procedures.

Table 1. Quality labelling initiatives and procedures

Procedure	Initiatives
Code of conduct	<i>Internet Health Coalition</i> <i>American Medical Association</i> <i>e-Europe</i> <i>Hi-Ethics</i>
Quality label	<i>Health on the Net Code</i> <i>Japan Internet Medical Association Mark (JIMA Mark)</i>
User guidance	<i>DISCERN</i> <i>Net Scoring</i>
Filtering tools	<i>OMNI</i> <i>CISMeF</i>
Third parties certification (trustmark/logo)	<i>MedCIRCLE</i> <i>Health on the Net Code</i> <i>URAC</i> <i>Web Médica Acreditada (WMA)</i>

Codes of Conduct: are defined as sets of quality criteria that provide a list of recommendations for the development and content of websites.

Quality Label (logo): is displayed on screen and represents a commitment by a provider to implement or adhere to a code of conduct.

User Guidance: this system enables users to check if a site and its contents comply with certain standards by accessing a series of questions from a displayed logo.

Filtering Tools: applied manually or automatically, accept or reject sites of information based on preset criteria. These tools are based on the “gateway” approach to organising access to the Internet- that is, resources are selected for their quality and relevance to a particular audience.

Third Parties certification (Trustmark/Logo): quality and accreditation labels are logos or symbols awarded by a third party usually to inform consumers that a site provides information meeting current standards for content and form.

The main problems that these mechanisms face are related to the lack of machine processable labels that can be identified and parsed by search engines or web browsers, as well as the need for continuous review and control of already characterised (accredited or classified) web sites and the location of ones that have not been characterised yet, tasks that currently require a huge amount of human effort. WMA, as third-party accreditation system, for instance, periodically reviews manually the accredited web sites to renew the quality label. On the other hand, in AQuMED, as filtering and rating system, web site directories are periodically updated due to the addition of new sites and changes in the characterization of the already visited ones.

MedIEQ aims at the development of a labelling platform, called AQUA (from Assisting Quality Assessment), to assist the work of labelling experts, increasing in turn the number of labelled medical sites and improving their monitoring.

In the case of WMA [13], the application of the platform tools concerns the constant monitoring of already labelled medical web sites comparing newly extracted information from the site pages against the data stored in the MedIEQ database (extracted in previous runs) or the labelling operator database. Taking into account the steps of the WMA labelling process these will be supported by the AQUA labelling system in the following ways:

- Every time a new request arrives to WMA, AQUA is invoked in order to collect an initial set of data from the corresponding web site. The type of data collected (they will vary according to the request type) will be stored in a separate database in order to be used by the WMA standing committee.
- After the site owner informs WMA that any committee recommendations have been implemented, the labelling system is invoked to examine the corresponding updates. The system outcome is again stored in order to be used by the labelling experts in WMA, who will decide whether the specific site will be labelled or not.
- After the site gets the WMA label, the system will be invoked periodically to examine whether any changes occurred, in terms of the labelling criteria. Depending on the change, the system can alert WMA, thus facilitating the monitoring process.

In the case of AQuMED [14], the application of the platform tools concerns the identification of new medical web sites, in specific thematic areas, their characterization, the filtering of some of them based on their characterization, and their organization into web directories. Taking into account the steps of the AQuMED labelling process, these will be supported by the labelling systems in the following ways:

- A focused web crawler will be trained to locate medical web sites for specific subjects.
- Every time a new web site is retrieved, the labelling system will examine it against AQuMED criteria and store the data collected in a data base separately from the data base storing the meta-data of the AQuMED web directories.
- In case the labelling system has to re-examine an already characterized web site, it checks first whether the previously collected meta-data are still valid

and in case changes occur it updates the data collected in the database, alerting the labelling expert.

- The sites that do not meet certain criteria are filtered and their data are stored separately in order to be examined by the labelling expert who will take the final decision on adding, excluding or withdrawing a site from the directory.
- The labelling system operates periodically in order to locate new web sites or update the data on existing ones.

3. Design Requirements for the Labels Management Toolkit

3.1. Objectives

The LAM toolkit, when finished, should become the one stop shop solution for labels management for the MedIEQ AQUA system. LAM users should be able to accomplish with LAM various label-related task.

The functionalities that LAM will be providing should make it easy for the users to create, update and delete labels, to monitor changes in labels and properly import and export labels to the system. All these should be available via a user friendly web interface integrated in the AQUA system look and feel and be accessible from it.

LAM should also be able to interact with other systems that might need its functionalities, in our case other toolkits from within the AQUA system. LAM needs to be as independent as possible from different labelling models. Its adaptation to model changes should take place with the minimal reprogramming effort and if possible via an administration user interface.

3.2. LAM in the AQUA Process

LAM is a crucial part of the AQUA process since it will provide the functionalities necessary for the RDF Content Labels themselves to be created. Its modules will be either called from other AQUA modules for creating the RDF labels semi-automatically by accepting their data extracted from web resources or will be used by human labelling experts for manual creation/editing of labels.

3.3. User Roles and Responsibilities

Two types of user agents will be interacting with LAM, humans through the user interface and S/W tools through well defined programming APIs. The latter case is the AQUA System's modules that take advantage of certain functionalities from LAM, so only one role is enough.

For the former case though we need to define different Roles and Responsibilities for users. Two types of users need to be defined: the Labelling Expert and the LAM Administrator. The labelling expert will be able to create and manage his/her own labels. Managing labels includes editing existing labels, saving labels in the version control system in order to track changes history, uploading and validating new labels and finally comparing label contents. The LAM Administrator is provided with the same operations as the Labelling Expert plus a few more related to administration of label creation, label validation, label model management and assignment to users.

3.4. Functional Requirements

The functional requirements that LAM should implement are the following:

- Label generation: to create new labels
- Label import/export: import existing labels to the system, export an RDF/XML serialization of newly created ones
- Label validation: 3-level validation of the label file to ensure that it can be

- processed by the system
- Label editing: edit existing labels to create new ones faster, or new versions of existing ones
 - Label versioning: a system that lets users track the evolution of labels
 - Label sharing among users: allow multiple users to work on a single label
 - Label model adaptation: change the model the labels were created

3.5. Non Functional Requirements

The implementation of the system and the specification of functional requirements are always driven by non-functional requirements that specify general properties that we want the system to possess from a generic conceptual, architectural and user perspective.

The system can be easily **Internationalized** (user interface and multilingual content handling) since it is integrated in the AQUA Internationalization systems, that guarantees User Interface message independence.

The system is **Portable** because it is developed using Standards Compliant technologies implemented by several Independent software vendors and thus it can be run in any Java EE compliant platform.

The system is **Extensible** by letting the Administrator add new RDF Vocabularies on the one hand and on the other by being able to use new services deployed on the container for other RDF models.

LAM can be easily packaged and used independently of the AQUA system to create RDF Content Labels and/or extended with other labelling models, which makes it **Reusable**.

4. Labels Management Toolkit modules

For each of the LAM module, a technical overview is provided, along with screenshots, usage scenarios, interface descriptions, input and output actions expected from users.

4.1. *Label Generation*

Label generation is accomplished either manually by the user or semi-automatically by the AQUA system hooks on LAM. The user has to fill a web form where he/she can assign values to several properties needed to have a proper label. The user can also inspect an automatically filled label generated after the processing of a web resource using AQUA.

The user interface for creating a label is a web-based form with input boxes, single and multiple select boxes, links and buttons. It is split in 4 distinct areas. The top three reflect the three logical parts of an RDF-CL [11] model-based content label and should be filled by the user. The last one, in the bottom of the page, arranges the serialization format of the RDF label and whether the label should be saved as a new version of an existing label or as a new one. This last part also gives the chance to the user to download a serialization of the label in the format he/she selects in a text file.

The first part of the label creation form asks from the user to fill the input boxes with label metadata. The fields to be filled hold data for identifying who created the label, when the label was created and altered and who is accountable for using certain RDF vocabularies.

The next part lets the user constrain the application of the label to certain hosts by explicitly declaring the host URIs or by adding regular expressions that properly identify them. Multiple hosts can be defined. Regular expressions for more fine grained addressing can be defined as well. These definitions can be combined via Union and Intersection and thus create rules that link different parts of a web resource with different labels.

The last part of the first three is where the label properties are assigned values. The label properties are the actual descriptors of a web resource. A set of label descriptors can be linked with a set of host restrictions defined in the second part. Related properties are grouped to help the user when filling them.

Once the user has filled the label metadata, restrictions and properties, he/she can save the label. There is a notification field that informs the user if the label exists in the system already and its changes are tracked from the version control system. In this case the user can save the label as a revision of an existing label. If this is not the case, the user just selects to save the label. In both cases the user has the option to download an RDF/XML serialization of the label to attach to the web resource he/she prefers.

My account

- [Edit my account](#)
- [Logout](#)

Quality labelling

- [My quality labels](#)
- [Labels Management](#)
- [Tasks management](#)
- [Alerts management](#)
- [Formation of corpore](#)
- [Linguistic resources](#)
- [The quality criteria](#)

The AQUA system

- [About AQUA](#)
- [Contact the system administrators](#)

Label Metadata

Organization Name:

Mailbox address:

Homepage:

Issued date: Date format : DD/MM/YYYY

Modification date: Date format : DD/MM/YYYY

Authority for:

Label Restrictions

Host Edit Host Restriction

Labels Properties

[Services](#) [Content Creation](#)

Property	Value
Target audience	<input type="text"/>
Virtual consultation	<input type="text"/>
Other seal	<input type="text"/>
Resource language	<input type="text"/>
Last Update	<input type="text"/>

Save Label

RDF/XML

Figure 1: Label Creation - Editing Page

4.2. Label Uploading

For the case that the user already has a label's file that conforms to the LAM label model he/she can upload it to the system and if valid, then take advantage of all the system's services. The labels are inserted to the system and connected with the web resource they define.

A simple form interface has been created. The user opens a file chooser to select the local file he/she wishes to upload. When the upload button is clicked the label file is transferred to LAM which validates the label for proper serialization syntax. It is checked if all the technology constraints that support the label model are followed

and if the uploaded file is indeed a label's file following the current labelling model (at the time of this deliverable RDF-CL). More on labels' validation in the next section. If the label passes successfully the three validation stages it is saved in the system in the user's workspace.

The screenshot displays the MedIEQ AQUA web interface. At the top left is the logo 'MedIEQ AQUA' with the tagline 'Assisting Quality Assessment system'. A 'Go to MedIEQ.org' link is at the top right. The main header features a blue cross icon and the text 'MedIEQ: Quality Labelling of Medical Web Content using Multilingual Information Extraction'. Below this, there are navigation menus for 'My account' (with links for 'Edit my account' and 'Logout') and 'Quality labelling' (with links for 'My quality labels', 'Labels Management', 'Tasks management', and 'Alerts management'). The central area contains an upload form with a text input field, a 'Browse...' button, a file type dropdown set to 'RDF/XML', an 'Upload' button, and an 'Edit' button. A message above the form reads: 'Please select a file to upload and a proper filetype. Then click the Upload button to successfully upload and edit it. Make sure you upload only RDF/XML files. Soon we will add support for N3 and Turtle notation.' A footer bar at the bottom right contains the text 'xhtmlcss cc 508 aaa'.

Figure 2: LAM upload label form

4.3. Label Validation

Label validation is an important service for LAM. It is executed when a label is uploaded to the system by an external source (e.g. user local directory). That label might not be generated by the AQUA system, so when a user uploads it, we need to make sure that it is a valid RDF serialization that contains RDF-CL (or any other model we define) labels. Otherwise we might not be able to process the label within our system. It might also pose security and reliability risks for the AQUA system in general, since someone might upload content that causes our system to crash or not be able to service other clients.

The label validation does not provide any user interface controls such as buttons, selection boxes etc, since it is called automatically while uploading a file. Validation results though are presented to the user in a textbox at the label uploading page.

Validation occurs in three stages, each one checking different aspects of the RDF Content Label instance. The first stage validates that the RDF serialization is correct. For example, if we are given an RDF/XML serialization of the label it makes sure that the document is first and above all valid XML. Then, control passes to semantic validation, where each of the Classes and Properties and other RDF elements instantiated in the labels file are checked against their RDF Schema declarations to find Domain/Range inconsistencies and type value assignments to Classes and

Properties. The last stage checks the RDF model generated by the uploaded file for the existence of certain structures that could identify it as being an RDF Label. If any single validation stage fails, the validation process is terminated and results are reported to the user. If all three stages succeed, then the uploaded file is saved in the system.

4.4. Label Editing

The label editing interface is the same with the label creation. The user can select a label to edit from the label listing and the label editing form will appear with all the fields filled with data. The user can change any metadata, label or host value, or add more label properties and host restrictions.

When the user is ready, he/she can select to save the label as a new version of the one he selected to edit and thus keep the label under the version control system that allows him/her to track the changes. There is always the option to save the edited label as a completely new one. The option to download the label is provided as well.

4.5. Label Versioning

We have created a Label versioning system (lvs). The lvs module lets us track the history of labels and consequently the web resources they characterise as well. Using this system, we can track the changes in each label after an AQUA system run and highlight the differences among versions.

Another benefit of versioning is the efficient storage and management of the RDF labels, since only the latest version of the label will be saved in the file system and all the previous ones will be saved as patches. When the user will ask to review an older version, the reverse patch application process will create the version the user asked for.

4.5.1 The versioning cycle

We will give a description of the process that we expect someone to employ in order to properly version labels.

The first step is to add a newly created label to the versioning system by using the **Add** action. The web form that is displayed asks for a user comment on the label he/she wishes to add. These comments will be helpful for the user when, at a later stage, he/she will go back to track the history and the changes made to a label over several versions. These comments, along with information about when a certain label version was added or changed will be later retrieved using the **Info** action. After the label is added to the versioning system, the user will be asked to use the **Commit** action every time he/she makes a new edit in the label via the label editing interface. Each new **Commit** or **Add** creates a new revision number for the system in general. Using the **Delete** action, this leads the versioning system to delete the label and its history. However, the label is not deleted from the Aqua system. It is still there and the user can use (edit/download/diff) the last saved version.

There might be a case that the user wants the last version of a label to be updated to an older revision. This is when the **Update** action will be called. Finally the **Diff** action can be used when the user wants to see the differences between two versions of the

same label or 2 certain versions of different labels.

4.5.2 Versioning Actions on the User Interface

The following list explains in more detail the actions that are available to the versioning system's user interface:

- Add: when a new label is created, or added to the system after an upload, the user can add it for version control.
- Commit: a commit action saves the new label version and creates a diff with the previous version.
- Info: information entered by the user before committing a revision.
- Delete: delete all versioning history about a label from the repository.
- Update: update the working copy of a label to an older version.
- Diff: the differences between two label versions.

4.6. Module Administration

Module administration interface lets the administrator change several system options on the fly while the system works. The module administration user interface consists of pages only accessed by the administrator. The options the administrator can change and their mapping to web pages follows in the next table

label properties <ul style="list-style-type: none"> • add/remove terms from RDF Vocabularies to be used as label properties • select which RDF terms to use as properties in labels. • organize properties in groups 	admin/lam_manage_terms.xhtml
Add / Remove RDF vocabularies	admin/lam_manage_vocabulary.xhtml
File uploading directory	admin/lam_upload.xhtml
Assign labels to users	admin/lam_labels_users.xhtml
Select model for creating labels (currently only RDF-CL)	admin/lam_model.xhtml
Version control options <ul style="list-style-type: none"> • manage the repository 	admin/lam_versioning.xhtml
Label validation properties	admin/lam_validation.xhtml

MedIEQ
AQUA
Assisting Quality Assessment system

[Go to MedIEQ.org](#)

MedIEQ: Quality Labelling of
Medical Web Content using
Multilingual Information
Extraction

MedIEQ

Administration options

My account
→ [Edit my account](#)
→ [Logout](#)

Quality labelling
→ [My quality labels](#)
→ [Labels Management](#)
→ [Tasks management](#)
→ [Alerts management](#)
→ [Formation of corpora](#)

- [Add Remove Terms](#)
- [Add Remove Vocabulary](#)
- [Select terms for properties](#)
- [Validation Options](#)
- [Assign labels to users](#)
- [Select label model](#)

xhtml css cc 508 aaa

Figure 3: LAM Administration Options page

5. Labels Management Toolkit: Development framework

5.1. LAM design in general

The Labels Management toolkit is a web application developed using JBoss Seam application framework [2]. JBoss Seam itself is built using Java Server Faces technology [3] for web page templating, rendering and navigation control. EJB3 session and entity beans are used for handling business logic, application logic and mapping the domain to entities used in our program, along with extensions for providing higher level control for session management, input validation, transactions, navigation, etc.

Our aim was to take advantage of the framework and the design patterns it supports to create robust, quality code that does not couple application semantics with user interface so as to easily and independently extend and/or change each part when necessary.

We have developed jsf web pages that contain only presentation tags. Navigation Control and actions from web page controls (such as buttons, links, etc.) is handled with EJB3 Stateful and Stateless session beans and a configuration file that maps web page files with aliases and URIs used in the program internally. The EJB3 Stateful and Stateless session beans encapsulate application logic and control the actions the user initiates at the JSF page user interface level.

Each EJB3 bean aggregates classes for handling and manipulating RDF models, saving to the database domain entities, getting input from user submitted JSF forms, changing that input and preparing it for display back again to the JSF pages.

5.2. Configuration

Using an application server and web development frameworks [1] relieves us from some work that we would otherwise do by programming it explicitly. Instead we take advantage of resources and services that the application server makes available to us. To use them in our application, we need to properly configure them. The services that we use relate to navigation, file uploading and authorization for using certain parts of the application.

We explicitly define navigation rules and conditional navigation in the files `pages.xml`, `faces-config.xml` found in the `WEB-INF` directory. We use the rules defined there in our EJB3 Session beans.

Authorization is defined declaratively in the file `web.xml` also found in the `WEB-INF` directory. We define URI paths that each user role can access explicitly.

5.3. Database

A database schema supports LAM modules. It was designed to persist data related to:

- the RDF namespaces and elements (Classes, Properties, etc.) that the modules use;
- the label files that every user uploads/maintains/reviews/edits;
- the RDF versioning system, we needed a place to store the versions each user

commits for a label.

The database schema was created in Postgresql v8.1.4 database and is part of the overall AQUA schema. We access the database from our EJB3 Session beans through EJB3 entity beans that map the tables to Java classes. The EJB3 persistence manager called from Session beans is used to save, update, delete and retrieve data to the database through the proper mapping entity beans.

The database schema includes the following tables:

- **RDF Labels Files** saves the absolute filename saved on the disk and assigns it an id to be later used as a reference for the base file version in diffs.
- **RDF File Diffs** acts as a repository for the label version system. Each tuple saved in it will be holding the version number, the base file id this version refers to, the link to the actual diff/patch, the date it was committed, the user id that committed it and his/her comment.
- **RDF Files Users** links the users with RDF files. It is a many-to-many relation; each user can have many files to work and the opposite.
- **RDF NAMESPACES** holds the namespaces that identify different RDF Vocabularies and their unique names that will be used for reference in RDF instances.
- **RDF TERMS** is a placeholder for all the rdf terms selected from different vocabularies to be used in LAM.

5.4. EJB3 Session bean Actions, Entities and Services

Application control, logic and model mapping are implemented with EJB3 Session and Entity beans and JBoss JMX managed services. We tried to follow some design patterns that would let us build a flexible componentized system that would be easy to extend and configure.

EJB3 Entity beans are Java classes that their properties map and model (datatypes, constraints) directly to database tables. For each database table we have created a different Java class except for the **RDF Files Users** table that models a many-to-many relation.

EJB3 Session beans are the application controllers. Each user request, whether it is a button click or a url request is mapped to a method in an EJB3 Stateful or Stateless Session bean that handles it by executing some other Session bean method, sending or getting data from the database or dispatching the request to a Container hosted Service. For example, when the user selects a file to upload, the upload method is called from the UploadAction Stateless bean that uploads the data, creates an RDF model from them, then dispatches the validation task to another EJB3 Session bean and if the validation succeeds calls another method from UploadAction to save the data to the server.

Application Server (Container) deployed managed services, give us, on the one hand, maximum flexibility over controlling, changing and configuring application logic in a manner independent of the application control and presentation and, on the other

hand, let us enable and disable services and functionalities while the application is online. We have created a JMX based mbean service to be deployed on JBoss that handles the application logic related to the RDF label creation. This service consists of plain Java objects and some configuration files with metadata that let the Container identify and control them.

5.5. JSF Interface

LAM web user Interface has been built with MyFaces Java Server Faces technology and Java facelets. We created a page template which contained the page structure, navigation menus, logos for header and footer. Then each new page was simply a facelet included in the template. Each new facelet created contains only the jsf and xhtml elements needed for a certain task. For example the facelet for the file uploading contains only a file selection control, an upload button and placeholder for validation messages. This way we have taken advantage of a templating system to maximize flexibility for our system. Each facelet renders static and dynamic text messages and sends actions to EJB3 session beans (stateful and stateless) through jsf buttons and links.

Seam provides us with jsf components, Java annotations for configuring session beans and linking them with the jsf pages and finally a higher level application context that lets us manage and track user sessions and workspace easier. Using seam we can spread the execution context of a logical procedure over several jsf pages and session beans using only a few annotations whereas if we didn't have it we would have to use lower level basic session management provided by the web server.

5.6. Libs

LAM uses third party open source libraries for certain tasks. Some of these libraries are used out of the box, adapting perfectly to our interfaces, while some others needed some API changes to better fit our design.

We used the Jena Semantic Web Framework for RDF model creation, manipulation and storage. The RDF-util library provided us an implementation of RDF Molecules [9] based diff and patch functionality that we used to build the RDF versioning system. For Semantic RDF Validation we adapted to our needs the Validating RDF Parser (VRP) [7] and Jena third party package Eyeball [8].

As we mentioned above, LAM is based on the Seam web application framework [2]. Seam itself is based on Java Server Faces technology [3] which it extends and integrates with EJB3 technology [4]. JSF is a specification published by SUN Microsystems. We used Apache MyFaces [5] implementation of JSF and some custom JSF components from the apache tomahawk library [6].

6. Concluding remarks

The design of the Label Management toolkit (LAM) took into account, on the one hand, the requirements of the labelling authorities, WMA and AQuMed, and on the other hand, certain general requirements, such as internationalisation, reusability, portability and extensibility. We implemented LAM in a way that adheres to those requirements and eases any future development and extensions.

After closing a first developing iteration, the system is stable enough to be used on production for all the labelling tasks described. In the future, we expect feedback from the users to add more functionalities, especially on the versioning system which is something new in the labelling process. We also plan to add more labelling models as we actively participate in a W3C process for defining a new generic labelling model for web resources [10]. This will be easy to do since our system is independent of any labelling model.

Last we would like to stress the fact that LAM can also be distributed independently of the AQUA system.

References

- [1] <http://jboss.com/>
- [2] <http://jboss.com/products/seam>
- [3] <http://java.sun.com/javaee/javaserverfaces/>
- [4] <http://java.sun.com/products/ejb/docs.html>
- [5] <http://myfaces.apache.org/>
- [6] <http://myfaces.apache.org/tomahawk/>
- [7] <http://139.91.183.30:9090/RDF/VRP/>
- [8] <http://jena.sourceforge.net/Eyeball/>
- [9] Ding L. , Finin T. , Peng Y. , Pinheiro da Silva P. , McGuinness D.L. , Tracking RDF Graph Provenance using RDF Molecules , ISWC 2005
- [10] <http://www.w3.org/2005/Incubator/wcl>
- [11] RDF-CL model (<http://www.w3.org/2004/12/q/doc/content-labels-schema.htm>) will be refined to a new model by the forthcoming W3C POWDER working group (http://www.w3.org/2006/12/powder_charter.html)
- [12] <http://www.quatro-project.org/>
- [13] <http://wma.comb.es/home.php>. WMA: Web Mèdica Acreditada (ES), a health quality labelling agency and a MedIEQ partner.
- [14] <http://www.aquumed.de/>. AQUUMED: Agency for Quality in Medicine (DE), a health quality labelling agency and a MedIEQ partner.

Appendix I – RDF-Content Labels model

Content labels for the MedIEQ project are created using the RDF language. Not only do the labels use Classes and Properties from RDF Language based vocabularies but they also follow a model that defines how the Classes and Properties are linked together to create a Content Label. The label model that we follow is the RDF-CL. Apart from defining an RDF schema with Classes and Properties, RDF-CL also defines three logical parts in the label's structure. Each part contains information that supports the other two.

The three main parts of a label:

1. Label Metadata
2. Label Restrictions/Grouping
3. Label Properties

Label Metadata

This part holds information about who and when created the labels, when they were last modified, details about the label authority itself, the vocabulary it is responsible for. Terms will be used from dc, dcterms, foaf and label vocabularies.

Label Restrictions/Grouping

This part holds mappings of labels to hosts and more detailed rules for explicitly linking parts of websites or resources to labels. Also elements (Intersection, Union) for combining these rules will be available. A default label is defined for the given host restrictions. More labels can be defined and linked with more fine grained descriptions given by rules. The terms come from the label vocabulary.

Label Properties

Each label contains a set of label properties it links to from the Label Restrictions/Grouping part.

Many labels can be defined, each one containing different RDF elements that denote the properties that best describe the content of the resource we wish to label. Label properties will be expressed using the criteria from table 3.

Appendix II – MedIEQ RDF Schema

The first version of the MedIEQ vocabulary includes the criteria presented in the following table (for more details, see Deliverable D4.1).

Criteria	RDF Vocabulary Term	Value
1. Title	dc:title	String
2. Resource URI	label:hostRestriction	String
3. Responsible / Author	foaf:name	String
4. Contact details of the responsible / author	dc:creator foaf:organization foaf:mbox foaf:homepage	<ul style="list-style-type: none"> • Email address • Postal address • Telephone number
5. Last update	dcterms:modified	date
6. Topic / Keywords	dc:MESH	For instance using the Unified Medical Language System (UMLS)
7. Resource language(s)	dc:language	String for language and language code
8. Target audience	wma:target	Considered groups: <ul style="list-style-type: none"> • Professional • Non – professional <ul style="list-style-type: none"> • Adult • Children
9. Advertisement	quatro:ac	Boolean
10. Quality seal or third party program	wma: otherseals	Considered seals. Real seals: <ol style="list-style-type: none"> 1. WMA 2. HON Code 3. pWMC 4. URAC 5. Health Trust-e 6. Afgis Virtual seals ¹ : <ol style="list-style-type: none"> 7. AQUMED 8. OMNI 9. WHO ("Vaccine Safety Net")
11. Virtual consultation:	wma:virtcons	Boolean

Table 1: The first set of MedIEQ labelling criteria

¹ In MedIEQ we consider a website having a “virtual seal” when it is recommended by one of these three organizations through their respective health portals.

The labelling criteria already exist as properties in the RDF vocabularies shown in the table below. The left column shows the vocabulary name and the right the URI namespace that this vocabulary has been defined and all its terms can be found.

rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
Dc	http://purl.org/dc/elements/1.1/
dcterms	http://purl.org/dc/terms/
quatro	http://purl.org/quatro/elements/1.0/#
wma	http://wma.comb.es/rdf/vocabularyv01#
foaf	http://xmlns.com/foaf/0.1/#
label	http://www.w3.org/2004/12/q/contentlabel#

Table 2: RDF Vocabularies for MedIEQ labels

Some details about the vocabularies:

- foaf : Has classes and properties for describing people, the links between them and the things they create and do.
- rdf : Provides a number built-in types and properties for representing groups of resources and RDF statements, and capabilities for representing XML fragments as property values.
- rdfs : Used for specifying rdf vocabularies.
- dc : General metadata about content such as creation ,author etc.
- dcterms : Refinements and further encodings to dc.
- quatro : To ensure accuracy of information ,compliance with rules and legislation for e-business , operating the trust mark scheme.
- wma : Related to medical content.
- label : Allows groups of resources to be linked to a common classification , description and also rules and operators to combine groups with descriptions.