Abstract

In this report we introduce our Web-application, explain for who it's built, what it does and how we have made it and with wat technologies. We also mention what changes have been made from our initial planning.

Report for Edocol

Edin Dudojevic (0608206)
Etienne van Delden (0618959)
Anson van Rooij (0596312)

June 12, 2010

1 About Edocol

Our project, the web application called "Edocol", is an education cooperation website for documentation. The two main purposes are aiding students and teachers in

1. learning about (technical) topics of their interest
2. creating new, high quality documentation in a colaborative manner

The two mechanisms for this are be

a sharing and creating information and,

b finding people with the same interests/courses or with required expertise/position (for ex. possible tutors or instructors).

What Edocol is: Our web-app occupies a place between Wikipedia-like information sources and social or professional networking sites; it enables sharing information and helps education, but not anonymously or in one specific format, and it facilitates cooperation, but always with a focus on creating/improving documents and furthering one's education.

What Edocol is not: This app is not intended to be a complete portal, like Studyweb or Sakai, nor to be as hierarchically organized. It's not an official information source like OWinfo or TUe.nl. It's not an

evaluation or validation system like Peach, a place to share unfinished, internal work like a Subversion server, a project management system like Projexy, or an email client like Outlook.

Features:
1. Personal accounts
2. an RDF database of "documents" and users
3. following users

Personal accounts: these are used to keep track of who have access to Edocol and to prevent anyone from adding and editting documents. Only users of the system can create now documents and only the original creator of a document can edit it.
RDF database: An rdf database is used to easily add aditional functionality. For example, tagging could be added by making a module that handles the requests and adds new properties and fields to new entries. The new type can be declared by making an *(id, edocol:type, "tag")* entrie, and extra data that needs to be stored can be used by adding the triple *(id, edocol:"property-name", value)*.
Following users: When you follow a user, you can quickly go to that users personal page and see the documents that person has made. This helps the sharing and creation of information.

1.1 Changes from the original design

The idea and design of Edocol is still the same. Only our must-have features have been implemented, though work has been made on a commenting systemand tags, we were not able to implement those on time.

2 Technical solution

For the creation of Edocol, we have used the following frameworks and technologies:

- CherryPy[1] for the HTTP framework
- Mako[2] for a templating system
- RDFlib[3] as an rdf framework and database storage

- (SQLite3 for storage)
- All of the code for both front and back end is written in Python

Edocol is split into two parts; a front-end (`server.py`) and a back-end (`database.py` with the `documentmethods`, `usermethods` and `tagmethods` modules). The front end handles the web interface; dynamically create web-pages with the content delivered by the back-end. The back-end is used to create the rdf graph, the query the database and to store it to file. SQLite3 was used to store the rdf database, but a threading problem was encountered for which no solution has been found yet. The rdf database is now stored in a file.

2.1 Changes from the original design

Instead of using Redland[4] as the rdf framework, we are using RDFlib. Redland is designed for UNIX based systems. With no UNIX server available at our disposal, we had to look for a different framework that would on both UNIX and Windows based systems.

3 Use of semantic data

Data is added to the rdf graph in one of two types: either a user or a document. Both these types have properties, these can be the same like "name" or different like "email address" and "content", that can be set to a certain value. This data is then added as a triple *(type, property, value)*. When the front-end needs info, the properties and the value are sent. The front-end can then filter what is needed.

4 Discussion

Despite that our target audiance is the educational branch, edocol is not bound to be used that. Anyone could use it for any type of documentation, like software development, finances and medical data.
By using an rdf database (and with tagging), creating an RSS-feed from any data would be easy. A lott of rdf based websites are also using RSS feeds. A switch to the Redland framework would make it even easier, for one if it's

features is built-in support for RSS feeds.

What have we learned from this project? We have learned how to design and work with an rdf database. It helped us see the benefits of using rdf but also see the state in which development around rdf is. We had a lott of problems with the initial setup of any rdf framework and later on with actually using it. There are multiple projects that each has it's own unique feature-set, but each also has it's problems including: missing SPARQL libraries, bugs, missing documentation, no up to date documentation, runs only on UNIX based systems ...
We also learned how to create a website with dynamic content. Using templates and a seperate backend gave a real seperation between creating a lay-out for content and the actual content.
Working with python was great! Python is an easy to learn language with a lot of built-in features. While we had trouble with setting up rdflib, we learned a great deal about python and how it works (and on limits of our own operating systems).

References

[1] http://www.cherrypy.org/

[2] http://www.makotemplates.org/

[3] http://www.rdflib.net/

[4] http://www.librdf.org/