A student/teach collaboration web application

2010-03-03

Abstract

Our idea for a collaboration web application, targeted to both students and teachers.

1 The web application

Our project is an education cooperation website. The two main purposes will be aiding students and teachers in

1. learning about (technical) topics of their interest
2. organizing, and taking part in, courses

The two mechanisms for this will be

a sharing information and,

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

What this project is: Our web-app will occupy 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 networking and cooperation, but always with a focus on creating/improving documents and furthering one's education.

What this project 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. Must-have features: Personal accounts, an RDF database of objects ("documents"), some standard types/topics.
2. Should-have features: Personal pages, RSS feeds, egalitarian organization, commenting on documents, dynamic output.
3. Could-have features: An elaborate branching/updating/approval/authorship (not "ownership"!) system, "following" people

4. Would-have features: API, links to (external) email accounts, licensing notice support, user-controlled type merging

Searching: Ideally one could search the database by adding or subtracting categories in any order. For example: a user should be able to search by choosing everything related only to the course Databases, then, from among the results, choosing all documents of type summaries, and then, from the results of that, choosing everything authored or co-authored by person A, then see only old versions. The user should be able to find the same thing by first choosing "author: contains A", then "type: only summaries", then "versions: not current", and then "course: only Databases".

2 Tools

- Our web pages will be built by using *HTML* for the general structure, *CSS* for the layout and *Javascript/AJAX* for dynamic interaction.
- For our rdf storage we will use *Redland* and for the queries *SPARQL*. The choice for *Redland* is because of the recommendation in the book, but also because of the support for *RSS* and *Atom* feeds which are a "should have" on our priority list.
- The web application will be built in *Python* by using the HTTP framework *CherryPy* [**?**].
- For our web pages we will also use *Mako Templates* [**?**].

3 Discussion

For our database we have two demands:

- We want to store a lot of content
- It needs to be expandable in a modular fashion

What kind of content do we want to add? While investigating this, we stumbled upon a realization. The content we want to store is of a different kind for different groups of people. While teachers are only interested in serving *courses*, *course information* and *lectures*, students are interested in *summaries*, *discussions* and study tools. The italic words are the different types of content that may or may not be used in our web-app. But they all have a common structure: they have a name, the content itself and a certain type. We have visualized this general structure in Figure **??**.

For example, a student could make a summary (the type), called "Summary - 2II25" for the course "Web Technology" (the name of a related course) with a piece of text explaining a number of topics (a blank node, with relations to other content). Such a topic could be RDF (the name), which is a topic (the type) and has text with the actual information for the user.

A teacher would be interested in different content. He could add a course (a

type) called “Web Technology” (the name) with some content. This content could be course information (both name and type) with text explaining what the course is about. It could also have a lectures page (again both name and type) with lecture slides (some files).

Visually, these examples are shown in respectively Figure **??** and Figure **??**, and combined in Figure **??**.

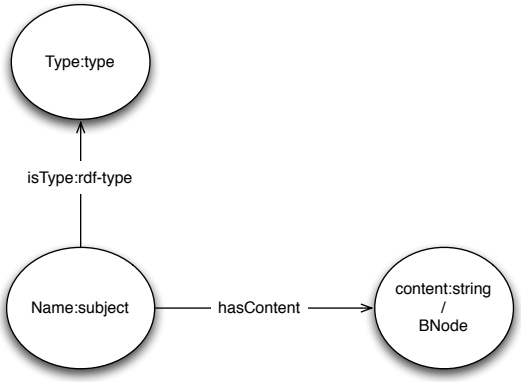

Figure 1: The structure of content

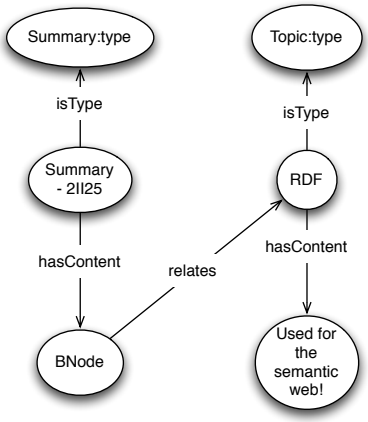

Figure 2: An example of content a student would add

Users can easily add their own content by relating it to an existing type. Users can also easily search by document type. Depending on how much we can implement in the given time, we can also assign properties to documents of a certain type. We could, for example, make it a property that a piece of content of a certain type may not have sub-content, only text, or the contrary; that it can only have other content. Using the current model, we can just add a property to a type, and all documents of that type will then have been changed.

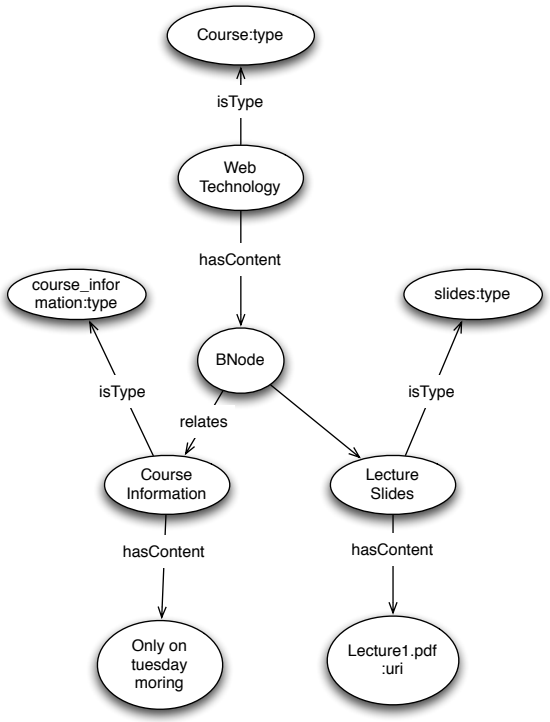

Figure 3: An example of the content a teacher would add

We also abstract from giving objects pre-defined types. A Dutch localization of our web-app could easily use the type "samenvatting", instead of "summary". But, even more powerful: the users can determine what kind of content they add. A summary, paragraph, question, course information, topic discussion, anything meaningful to the users can be added. We could somehow "merge" types which are essentially the same (such as "samenvatting" and "summary"), while remembering its different names.

Ownership can also be easily expressed in an RDF model. When ever new content is made, a relationship with the creator is made, making him the owner. Then, when someone creates a new "branch" of the object, the original creator can "approve" it, to label it as an updated version of the older object. This will help us to create the social networking aspect of our web-app. When someone "follows" a person, we could easily find all content created by a person and show the changed content.

RDF is a really powerful data-model to express relationships. By using RDF, we can relate all kinds of different data, and easily add new features by simple creating a new relationship. This makes RDF the perfect solution for our needs.

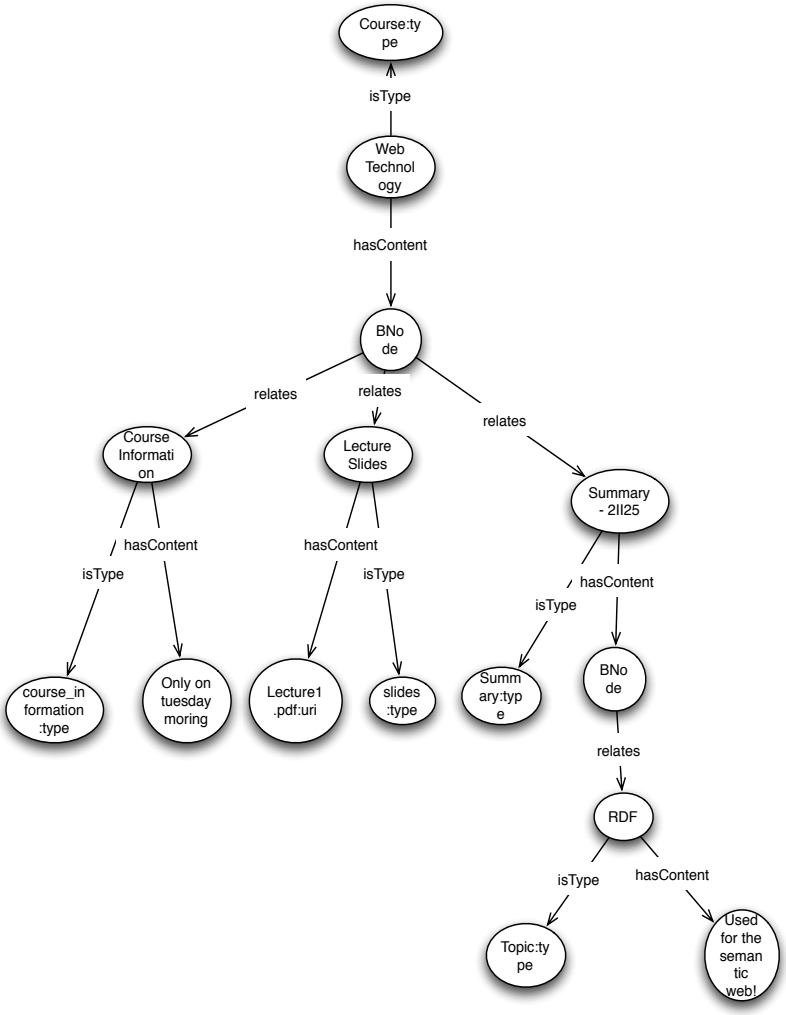

Figure 4: The RDF database, both examples combined

References

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

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