

Managing software artifacts on the Web with Labyrinth¹

Fabiano Cattaneo, Elisabetta Di Nitto, Alfonso Fuggetta, Luigi Lavazza, Giuseppe Valetto

CEFRIEL - Politecnico di Milano

Via Fucini, 2

I-20133 Milano (Italy)

Tel: +39-02-23954.1

Fax: +39-02-23954.254

E-mail: {cattaneo|valetto}@cefriel.it, {dinitto|fuggetta|lavazza}@elet.polimi.it

ABSTRACT

Software developers are increasingly exploiting the Web as a *document management system*. However, the Web has some limitations, since it is not aware of the structure and semantics associated to pieces of information (e.g., the fact that a document is a requirement specification) and of the semantics of relationships between pieces of information (e.g., the fact that a requirement specification document may be associated to some design specification document). In the Labyrinth project we enhance the capabilities of the Web as a document management system by means of a *semantic model* (called *schema*, in analogy with database schemas), which is associated to Web documents. This model is itself a Web document and can be accessed and navigated through a simple Web browser.

Keywords

Document management systems, software repositories, semantic models for the Web, Internet technologies.

1 INTRODUCTION

The simplicity and flexibility of the concepts underlying the Web has made it a successful –though limited– *document management system*. The main limitation of the Web as a document management system is its lack of knowledge about the semantics of the documents it stores. To manage large quantity of documents, it is essential to have visibility over a *semantic model* that describes the structure of these documents in terms of proper types and relationships among these types. This is especially true in the context of software development, where it is essential to have in mind the types of artifacts being produced (requirement specs, design specs, source code, test lists, ...) and the relationships among them (source code *implements* design specs, test lists *depend on* the requirement specs, ...). Such information can be exploited, for instance, to track the state of the development process (e.g., how many requirements have been considered in the

design phase) or to analyze the impact of a change of some artifact over the others.

Based on the above considerations, we have developed an environment called Labyrinth [1]. Labyrinth allows users to define a *schema* for a document base and to store it within a Web document. The schema defines the semantics of documents and relationships among them and is linked to the actual documents (collectively called *document base*). The schema and the meta-information layer can be accessed and browsed through standard Web browsers, thus enabling a semantic-based navigation of the document base. Labyrinth also provides a run-time infrastructure that manages the updates of the document base and monitors its consistency with the schema.

Our approach guarantees scalability: documents can be distributed everywhere, simply exploiting the basic features of the Internet. Also, it allows existing documents to be easily integrated in Labyrinth. In fact, any document can be part of a document base (i.e., the set of documents managed by Labyrinth) provided that its type and its relationships with other documents are described in the schema and that it can be accessed through an URL. For instance, documents stored in proprietary repositories that provide a Web interface can be linked to a Labyrinth document base. In this way a “super repository” offering a view over data stored in different tools is achieved.

2 THE DATAWEB

In Labyrinth a document base and the corresponding schema and meta-information layer define a *dataweb* (the term *dataweb* has been suggested to us by David Reese of Colorado University, Boulder).

The schema defines the structure of the document base. We have chosen to describe it through the traditional Entity/Relationship model whose expressive power is adequate to describe typing properties of documents and relationships among them. The evolution towards more sophisticated semantic models (e.g. object-oriented) and representations (e.g., through XML) is possible, while

¹This work was partly supported by MURST 40% funding programme, project Interdata.

retaining the general structure of Labyrinth.

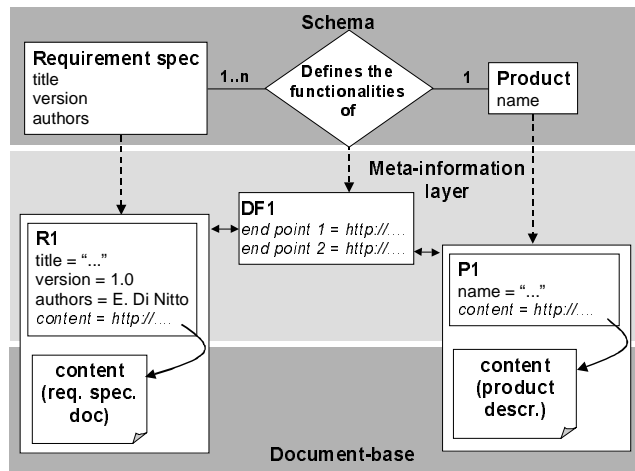


Figure 1. An example of a dataweb.

Documents have an *actual content* and are characterized by a *document description* that is structured according to the corresponding type definition given in the schema. The actual content is stored separately from the corresponding description and can be in any format (e.g., binary, ASCII, XML). Like the entities, the relationships defined in the schema do have an instantiation in the dataweb.

Document descriptions and relationship instances define the *meta-information layer* between the schema and the document base. They describe the characteristics of documents in terms of the semantics given in the corresponding model. The correspondence between elements in the schema (entities and relationships) and elements in the meta-information layer is explicitly maintained [1].

In summary, a dataweb is defined by a set of document contents and a number of HTML files that provide meta-information over such contents, thus enabling the definition of advanced navigation paths and query mechanisms. These mechanisms recall the ones provided by DBMSs but in a lightweight style.

3 HIGH LEVEL ARCHITECTURE

Figure 3 shows the Labyrinth architecture. A Web site, called the *Labyrinth home site*, stores the schema corresponding to a certain document base. The other sites contain fragments of the meta-information layer and of the document base.

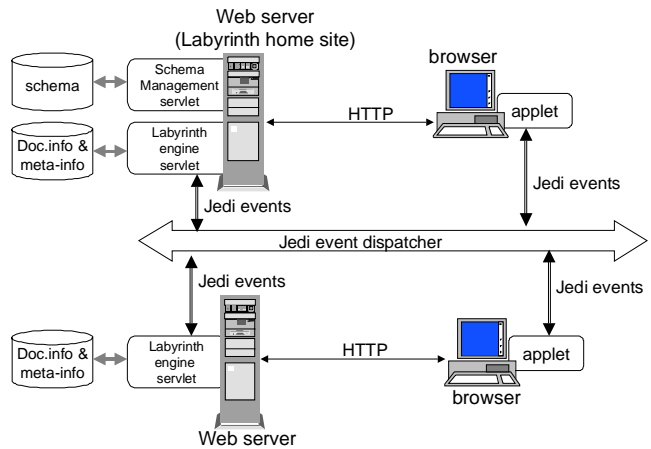


Figure 3. The Labyrinth high-level architecture.

Navigation of a Labyrinth document base is performed by exploiting the standard Web browser/server interaction paradigm (i.e. Labyrinth specific components are not involved if the user only browses the model and documents). Associated to each Web site there is a *Labyrinth engine* (implemented as a Java servlet) that controls all the update operations performed on the dataweb. It checks for coherence between types and instances, keeps track of possible inconsistencies (e.g., a document of type A should be associated to a document of type B but at the moment it is not), and ensures the atomicity of operations.

The JEDI event-based communication mechanism [2] has been adopted to notify users when some portion of the document base of their interest has been modified.

4 FEATURES OF LABYRINTH

The features of Labyrinth are illustrated through a case

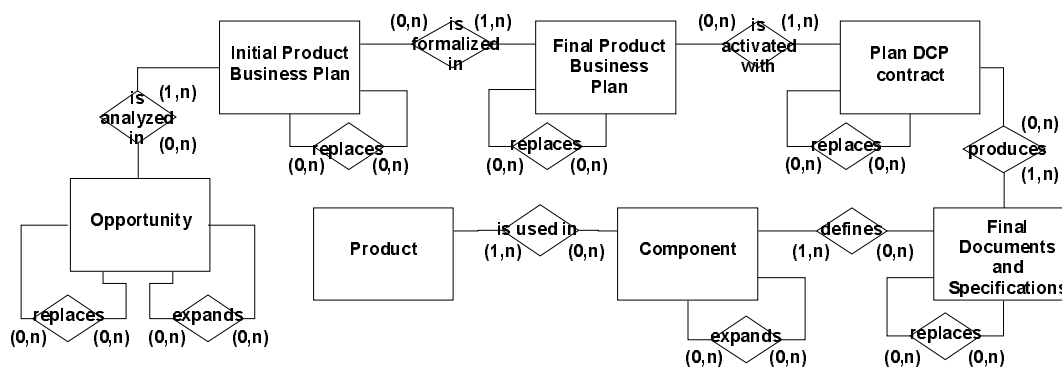


Figure 2. The schema used for the case study.

study we have carried out in an industrial context. A company develops and builds hardware devices with embedded software following an ISO 9000 certified development process. The company has an internal document repository based on Lotus Notes, where all the document contents are stored. The repository is composed of several Notes databases distributed worldwide. The company wants to use Labyrinth in order to build a semantic navigational and management model on top of the existing document base. In particular it is required that this model explicitly represents document versioning information and enables transparent access to documents regardless their physical location.

Figure 2 illustrates the schema as derived through the analysis of the existing documents and their relationships. All the instances of the entities shown in the figure are changed quite frequently during the development process. In the Notes repositories no explicit relationships are kept between different versions of the same document. In the Labyrinth schema an explicit “replaces” relationship has been defined, which allows contributors to the document base (document authors, mainly) to explicitly record that a document supersedes an old one and why.

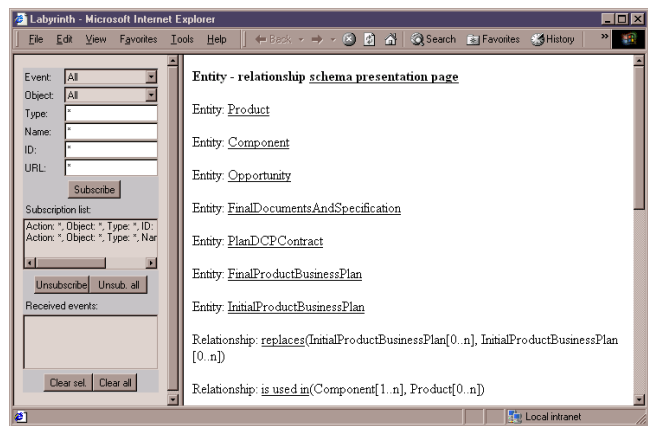


Figure 4: The schema presentation page.

Browsing and Populating the Document Base.

When the schema given in Figure 2 has been installed, a user can connect to the schema presentation page (see Figure 4) where all the entities and relationships belonging to the schema are linked to a list of the corresponding instances. As shown in Figure 4, in the current implementation the schema is displayed textually. We are working on its graphical representation.

From the schema presentation page, a user can navigate through the existing documents or he/she can request creation of a new entity or relationship instance. Figure 5 shows the form that is downloaded on the user’s browser when he/she requests to add a new document of type Opportunity (see the schema definition in Figure 2) to the document base. The form has to be filled in with the description associated with the document. The information to be provided corresponds to the attributes of the entity

according to the schema definition. Additional fields allow the user to select the Web server that will host the document description and the URL of the document content. Optionally, it is possible to have the Labyrinth system directly managing the storage of the document content. When the form is submitted to the Labyrinth system, it updates the dataweb by creating a Web page containing the document description and by linking it to the schema. Finally, if needed, it uploads the document content and stores it in the Web server file system.

As demonstrated by the example above, some operations on the dataweb require updates on different, possibly distributed files. The execution of these updates needs to be atomic and serializable. To ensure these properties we have adopted an approach based on the OpenGroup’s Distributed Transaction Processing concepts, where a manager coordinates the execution of transactions according to the two phase commit protocol. In the current implementation we use the Jini transaction manager (mahalo) [4]. The adoption of an ACID transactional approach in this case does not limit the scalability of the approach, since the number of involved parties is small. While the transaction is executing, users can still access the dataweb by exploiting the standard Web server mechanisms. If users perform operations that impact on the files involved in the transaction, these operations are serialized properly.

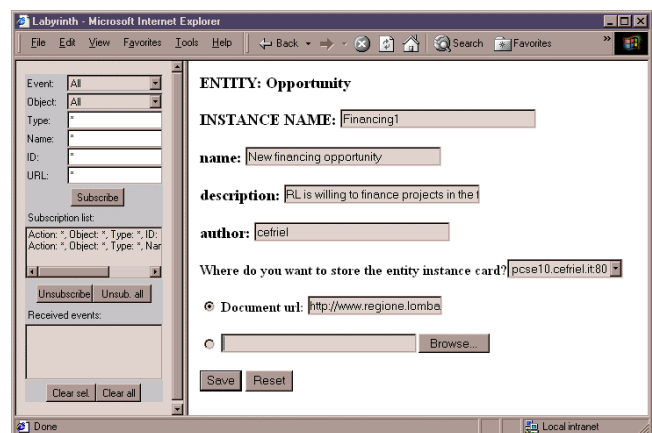


Figure 5. Entering meta-information about a document.

Handling Concurrent Accesses.

The Web, per se, does not provide any mechanism for guaranteeing that a user does not accidentally overwrite modifications performed by other users on a document. In some approaches (see for instance WEBDAV [3]) a locking mechanism is introduced. In Labyrinth we adopt a *lazy* approach where no explicit lock is applied to documents. Users download dataweb items on their machine and perform any modification on such a local copy. When they end with the modification, they submit the change to the system. At this point, the Labyrinth engine checks if the modification is *compatible* with the

current state of the dataweb, i.e., nobody else changed that same document in the meanwhile. If an incompatibility occurs, the Labyrinth engine does not apply the changes posted by the user arriving later, and requires him/her to merge his/her changes with the ones submitted by the previous user.

Handling Inconsistencies.

In principle, the document base should be kept consistent with the given schema: for instance, all the mandatory relationships for a newly uploaded document should be specified. However, in some cases maintaining such consistency is not feasible or convenient: e.g., a mandatory relationship cannot be established, since the correlated document will be released at a later time. In such cases, the user can leave the document base in an inconsistent state. Labyrinth will keep track of these inconsistencies marking the affected dataweb elements with a proper HTML tag and a textual description explaining the reason of the inconsistency (see Figure 6). Later on, as soon as the user provides the missing relationship, Labyrinth automatically detects that the inconsistency no longer exists, and transactionally replaces the tag with a proper link.

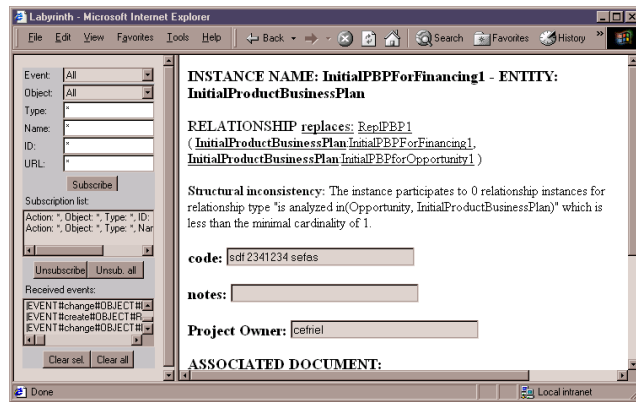


Figure 6. Dealing with inconsistencies.

Handling Notifications.

Labyrinth's notification mechanism complements the pull style approach typical of Web browsers. Labyrinth engines publish notifications about the occurrence of changes in some dataweb fragments using JEDI. On the user's side, each time a page describing a document or a relationship instance is downloaded, an applet (that is downloaded together with the page) automatically subscribes for notifications concerning the page. The user is thus informed as soon as someone else changes the page and can choose to have it automatically refreshed when this happens.

As shown in the left frame of Figure 6, another applet allows the user to express his/her interest in specific actions (e.g., in updates) concerning a particular entity/relationship type or even a single instance. This applet collects the notifications about such events emitted

from the Labyrinth engines and displays them. Examples of notifications that can be shown are: insertion in the document base of a new "Opportunity" document; insertion of a new "replaces" relationship for the "Final Documents and Specifications" entity; update of the description of a specific "Component" document.

5 DISCUSSION AND FUTURE WORK

Labyrinth combines a lightweight Web-based approach with an explicit, expressive and fully customizable schema. The existence of the schema makes Labyrinth suitable for application in several fields –such as software engineering– where it is important that documents conform to a predefined type, and relations (e.g., dependencies) are explicitly represented.

In the short term, Labyrinth will be integrated with a workflow management system and with several tools that will assist the user in defining the schema, in deploying datawebs on remote sites, and in defining policies for managing inconsistency. The modular servlet-based architecture of Labyrinth will also be exploited to add information search utilities.

A detailed presentation of Labyrinth features and architecture and a comparison with related work and approaches can be found on the CEFRIEL Web site at <http://www.cefriel.it/Se/Projects/Labyrinth>.

ACKNOWLEDGEMENTS

We would like to thank L. Aprile, D. Baroncelli, and L. Corradini who helped us in the development of the Labyrinth prototype. Special thanks to A. Agostoni, F. Perego, and E. Tracanella who have worked with us at the definition of the case study we have presented in this paper.

REFERENCES

- [1] F. Cattaneo, A. Fuggetta, L. Lavazza, G. Valetto, Labyrinth: Schema-based Distributed Document Management on the Web, CEFRIEL Technical report - RT 99002. <http://www.cefriel.it/Se/Projects/Labyrinth>
- [2] G. Cugola, E. Di Nitto, A. Fuggetta. Exploiting an Event-based Infrastructure to Develop Complex Distributed Systems. In *Proc. of the 20th International Conference on Software Engineering*, Kyoto, Japan, April 1998. <http://www.cefriel.it/se/Resource/resource.asp?ID=1>
- [3] IETF WEBDAV Working Group - World Wide Web Distributed Authoring and Versioning. <http://www.ics.uci.edu/~ejw/authoring/>
- [4] Sun Microsystems. Jini connection technology. <http://www.sun.com/jini>