# Managing Dynamic Repositories for Digital Content Components

André Santanchè and Claudia Bauzer Medeiros

Institute of Computing,
State University of Campinas – UNICAMP,
CP 6176, 13084-971 Campinas, SP, Brazil
{santanch,cmbm}@ic.unicamp.br

**Abstract.** The Semantic Web pursues interoperability at syntactic and semantic levels, to face the proliferation of data files with different purposes and representation formats. One challenge is how to represent such data, to allow users and applications to easily find, use and combine them. The paper proposes an infrastructure to meet those goals. The basis of the proposal is the notion of *digital content components* that extends the Software Engineering software component. The infrastructure offers tools to combine and extend these components, upon user request, managing them within dynamic repositories. The infrastructure adopt XML and RDF standards to foster interoperability, composition, adaptation and documentation of content data. This work was motivated by reuse needs observed in two specific application domains: education and agro-environmental planning.

## 1 Introduction

The reuse of digital content is an issue that is attracting increasing attention. In the context of this work, reuse can be defined as the practice of use an existing content object to build a new digital artifact using the object's content partial or totally [7, 13]. Reuse advantages include productivity improvement and cost reduction on development and maintenance. Furthermore, frequency of reuse may be a quality indicator, and content units designed for reuse can be quickly reconfigured.

On the other hand, there are several obstacles to supporting reuse; perhaps the most serious is the problem of proliferation of data, systems and users. Several directions are being followed towards solving these problems. One direction, which is investigated in this paper, is to exploit the advances in the area of software component reuse, from Software Engineering, combining them with database research on the Semantic Web and interoperability.

The Semantic Web [9] foresees a new generation of Web-based systems, where semantic descriptions of data and services will booster interoperability. In parallel, Software Engineering has reached a high level of maturity concerning reuse units, by developing the technology of software components. Our idea is to extend these principles, to comprise any digital content. From now on, this extended notion of component will be called *digital content component*; the term will be used in this paper to denote any kind of data e.g., pieces of software but also texts, audio, video, a result of a database query, and so forth.

W. Lindner et al. (Eds.): EDBT 2004 Workshops, LNCS 3268, pp. 66–77, 2004.

Software components are built by assembling code into a standard package that encapsulates implementation details. Well defined interfaces are associated with the package structure. These interfaces define how the component will be adopted and adapted into a new application, and how it will relate with other components and its execution environment.

By the same token, our proposal for a digital component structure involves the encapsulation of specific data representations into a package with a standard format, and public interfaces that support relationships among components.

Though the advantages of such generalization are evident, there remains the problem of putting it into practice. Thus, the paper is concerned with three main issues, having digital content reuse and interoperability in mind. The first issue concerns *establishing a model* to represent a digital content component, adopting interoperability standards preconized by the Semantic Web initiative. The second defines a *strategy to store and index* large volume of these components in a database. Finally it is necessary to *build a framework* to implement and manage content components. Our research has been motivated by reuse needs experienced by work developed at the State University of Campinas, Brazil (UNICAMP) in two application domains: education and agro-environmental planning.

The remainder of this text is organized as follows. Section 2 introduces related work. Section 3 presents the proposed digital content component model and discusses storage and implementation considerations. Finally, Section 4 presents concluding remarks and the present stage of this work.

## 2 Related Work

Related work involves research on software components and reusability in Software Engineering, and standards for digital content and interoperability in the Semantic Web. We discuss some attempts in these directions, starting from the software notion of component and evolving to the database concept of repository.

### 2.1 Packages

Content reuse can be enhanced via assembly into a standard package for distribution. In our work, *package* is defined as a structure that delimitates, organizes and describes one or more pieces of digital content suitable to reuse.

This packaging approach has already been adopted in the educational domains. The main example is the IMS Content Packaging Information Model [19], and based on that, the SCORM – Sharable Content Object Reference Model [5], a proposal to structure and distribute educational content in a package content form. Both aim at reusability, interoperability, location and adaptation facilities. They use XML to describe a hierarchical structure of each content package and their respective educational metadata. Metadata representation follows an IEEE standard for educational metadata, named LOM – Learning Object Metadata [8]. LOM has been coded in XML [23], with subsequent studies to represent LOM in RDF [10].

The adoption of open standards for component package structure and interface definition is a key approach to achieve interoperability. Within the Semantic Web context, XML and RDF are complementary standards, adequate to syntactic and semantic interoperability, respectively [4].

### 2.2 From Packages to Components

The package structure has limited capabilities. An evolution of this concept is achieved when packages are designed not only to transport content via a standard container, but to be connected with other packages and interact with them and their environment. This corresponds to the concept of *component*.

The component concept is often associated with the software component concept, which deals with software code. There is no agreement on the definition of the software component concept, though definitions are closely related [2].

Like a package, a component is designed to be a unit of independent deployment. There is a clear division between the content (encapsulated software implementation) and some external structure where this content is encapsulated.

However, components have higher specialized external structures, which publish component functionality by explicit contractual interfaces. To interact and work together, component structure and interface follow a model that specifies design constraints.

### 2.3 Component Repositories

Component reuse can only become effective with adoption of a structure to store and manage components in an efficient way. Many research initiatives concern component storage and retrieval, especially on techniques to index components. Indexation can be content-based or structure-based.

Prieto-Díaz [11] confronts two classification principles borrowed from library science, to apply in component indexation. First, the *enumerative* method uses a classification tree to organize components in categories and sub-categories. Second, the *faceted* method describes components by a set of attributes (named facets); each facet is specified by setting a pertinent term value (comparable to attribute value, but restricted on a list of possible values). Another option is using ontologies [20] to represent domain knowledge associated with components.

Still another indexation solution uses component structure. The basic method relies on component signature match [24], but it can be refined by a formal specification of component behavior, used as a basis to behavior match [25].

## 3 A Proposal for Digital Content Components

### 3.1 Components' Life Cycles

Any digital component infrastructure must support the entire cycle of component production, storage, retrieval and use – see Fig. 1. Production comprises the well known software component production process, which we propose to extend to any piece of data

the user wants to share. Such components can be assembled automatically, or guided by the user, through a tool named *packager*. The figure shows packager modules that encapsulate distinct kinds of content components – spreadsheets, workflows, maps, etc. A packager takes the form of a plugin attached to a software, which deals with the content, or an independent module specialized to process some file formats. Components are stored in a dynamic repository, and managed by a repository manager, which is accessed by users to retrieve and combine them. Again in the figure, a developer uses an authoring tool to compose distinct components – workflow, maps and workflow runtime engine – into an executable unit to produce a map. This specific example reproduces a scenario we deal with in UNICAMP, but without support of content components. In this scenario, experts specify workflows to generate maps for environmental planning based on combining several kinds of data. These workflows can then be run to produce distinct kinds of environmental plans.

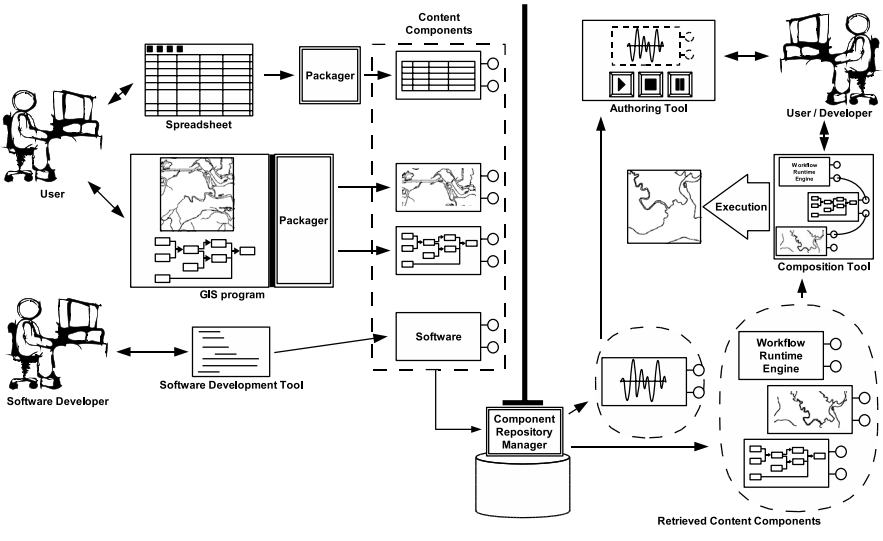


**Fig. 1.** Diagram of content component cycle for production/storage/use

The starting point for our work is the Anima project, an infrastructure for software components [14] developed by the first author. Anima is being used to create educational tools in several schools in the city of Salvador, Brazil. Anima comprises the complete cycle illustrated in Fig. 1, but restricted to software components and without database support. It provides support to building applications via component composition and uses RDF to represent component packages, including interface specification and component metadata. This allows applications to deal with software components implemented in different languages. An Anima application can be represented via a network of components whose configurations are stored in an XML file. This file can be dynamically converted into applications implemented in specific languages throughout XSLT sheets.

Moreover, Anima provides support to component execution and intercommunication via an XML based protocol.

With this background in mind, our research considers three aspects of content components: representation, storage/retrieval management and use.

### 3.2 Component Representation

**Content Component Structure**

A component's structure is defined to be a unit, composed of four distinct parts: (i) The content itself, in its original format; (ii) an XML specification of the internal structure used for component organization, based on SCORM [5], which allows a hierarchical description of components and sub-components; (iii) an RDF specification of component interfaces; (iv) RDF metadata to describe functionality, applicability, use restrictions, etc. Components can be recursively constructed from composition of other components, each of which in turn is structured by the same four parts. It is important to point out that XML has been proposed for aspect (ii), whereas all others are to be specified in RDF. These choices are based in the following criteria.

The internal organization structure follows a schema whose format and interpretation applies to all components. XML is a better choice in this case, where it acts as structuring element. Additionally, it allows inclusion of links to external pieces, allowing reuse by reference, as explained in [7]. External referenced entities include data generated by remote units, Web services, etc.

RDF is the choice to represent the interface and metadata, since descriptions and taxonomies are involved. RDF descriptions will promote straightforward indexation and component management support, and can be extended to represent ontologies with OWL [18]. RDF metadata with digital signature can ensure component provenance, and thus be used to control component quality based on the source.

Figure 2 shows a partial example of a component specification for encapsulating rainfall data. The content part stores a list of geographic positions for the rainfall stations coded in XML. The metadata part represents component's title and category. Specific terms employed – such as "XML Rainfall Station List" – are extracted from domain-dependent ontologies (e.g., see [12]).

The interface part shows three inputs and two outputs. All inputs are categorized as simple messages without parameters ("single"). Outputs describe in RDF how the contents are formatted (type arrow) for a given ontological context within a domain ontology (category arrow). This shows output should always contemplate not only syntactic information (i.e., XML schema), but also semantics (ontology terms).

This four-part structure for component representation, which extends the Software Engineering concept of software component to any kind of content, and promotes interoperability, is an important contribution of this work. Related work deals with some of these aspects in an isolated form, without an integration perspective.

**Categories of Content Components**

We differentiate between two kinds of component – process and passive components. A process component is a specific kind of digital content component. It encapsulates any kind of process description (sequences of instructions or plans) that can be executed by

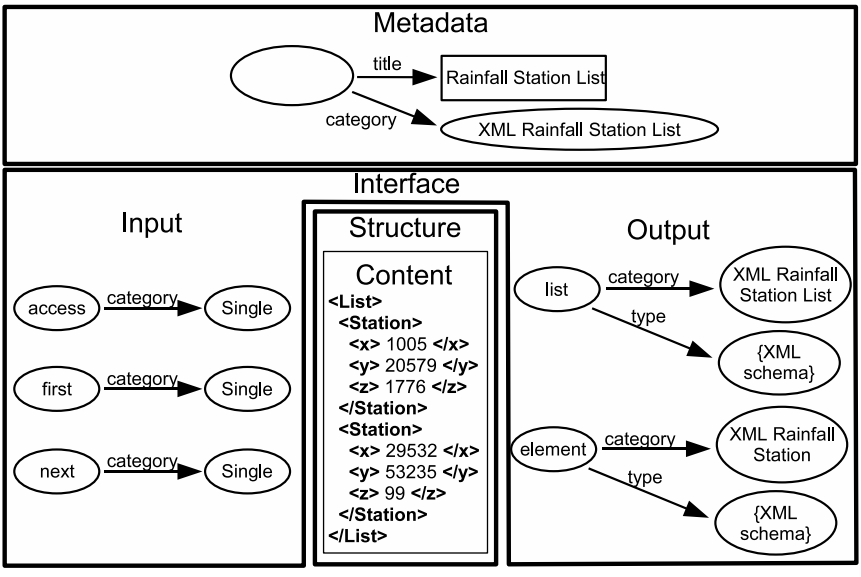


**Fig. 2.** Diagram of content component structure

a computer. Therefore, they usually define an input interface, and their results change according to different input values. The component of Fig. 2 is a non-process (passive) component, and contains data that can be used by a process component. A passive component's interface matches the input interfaces of process components enabled to use its contents.

In order to illustrate content component categories, we will borrow an example from scientific applications. In this context, scientists are interested not only in reusing results, but in sharing the whole process of experiment development. This originated the notion of scientific workflows (e.g. [1]), to specify and record experiments; this allows, among others, experiment reproducibility and therefore reuse. The WOODSS system [15], developed by us, follows this approach: it enables the capture of activities in agro-environmental planning to be stored as scientific workflows, which can be later edited, composed and re-executed.

WOODSS' users manage two main kinds of file: maps and workflows. The same workflow can be executed using different input maps. Workflows are an example of our process components, whereas maps are typically passive components. In our analogy, a "workflow component" can be linked to different passive "map components". Furthermore, one may envisage defining interfaces to workflows, in RDF, that will impose conditions on input files – e.g., indicating maps that can be acceptable as input.

Any digital content can potentially be packed in a component. The two requisites for packaging are: existence of an appropriate packager for its format and, for process components, existence of a runtime module enabled to run it. By the same token, two or more components can be attached and packed into a higher level component. For instance, consider a workflow component that receives two inputs: a map and a value $v$. It applies a sequence of filters to the map, based on value $v$ – and generates another map

as output. These components can be stored separately or composed into a more complex component that will deal with the specific map, and accept only input *v*.

Most process components cannot be executed directly, due to their need for interpreters or runtime modules, not embedded into their environment. In such cases, complementary attached components can perform this task. These complementary components are named *companion components*. Process components, moreover serve as companions to passive components, i.e., a passive component needs some sort of code to be processed.

The choice of the appropriate companion component to be associated to another (passive or process) component is determined by the application manager, and is influenced by the context. This allows dynamic component companion binding. Since this binding is defined by the application manager, it is transparent to end users. This allows a homogeneous treatment of passive and active components from the user's perspective.

**Using the Components**

Content components can be used in many ways. Three forms of expected use are: component insertion, content insertion and application construction.

In the first approach, components are inserted into documents or multimedia productions as content pieces. This use of components can be compared with the insertion of DDE/OLE objects into Windows documents, or insertion of embedded objects (such as Java applets) into Web pages. The components inserted are commonly passive components, or process components attached to passive components. For instance, a map component attached to a workflow component can be inserted in a scientific report.

The content insertion approach is similar to component insertion. However, the client unpacks component content and only this content is inserted into document or multimedia production, without component structure and metadata. In this approach, content components are used like content packages.

In the third usage form – application construction – components are used as basic blocks to construct applications. Here, just like in software component composition, an application is built from a network of interconnected components, following an architectural style [17].

This style guides digital content component usage and composition principles for construction of applications. A software architecture using components defines a configuration which involves components and connectors to bind components together. Our principles combine the Anima model for component composition with some aspects of the architectural style of C2 [22] – a message-based connection style for GUI software.

In our architecture each component is an independently executing peer, with its own state and thread of control. Components can only communicate asynchronously with other components via connectors – they can not use other forms of direct communication. Connectors transport messages; messages must offer support for at least XML, but can support another formats too (especially for performance purposes).

A message contains label, type and parameters represented in XML. The types' formats are defined by an XML schema and described by an associated RDF resource. The same schemes, taxonomies and descriptions are used in component interface description.

Figure 3 presents an example which combines process and passive components into an application intended to display informations about a set of rainfall stations. A modified version of the component illustrated in Fig. 2 will be used here, with some additional informations about the stations.

To compose components together the application adopts an architectural style named publish/subscribe [6]. Publish/subscribe style is based on messages produced by components and distributed by a message manager. Components subscribe to events they are interested in. Events are signalled by messages. In the example of Fig. 3, five components are subscribing to events (represented by a large arrow).

Publishers are components that raise events by publishing a message that is forwarded to every component that subscribes to that event. In the example, there are two visual software components (at the left side) that play a role of buttons and publish messages when the user clicks on them. The messages ("access" and "next") are dispatched to XML Rainfall Station List component, which subscribed to these events.

The "access" message induces the component to produce a "list" output message, which contains the list of all rainfall stations. Two components containing XSL stylesheets receive this message and convert it to a report and to a graphical representation of the rainfall station positions. The message "next" contains one rainfall station (the next of the sequence) and follows an equivalent path. This composition results in a browser of rainfall station lists.

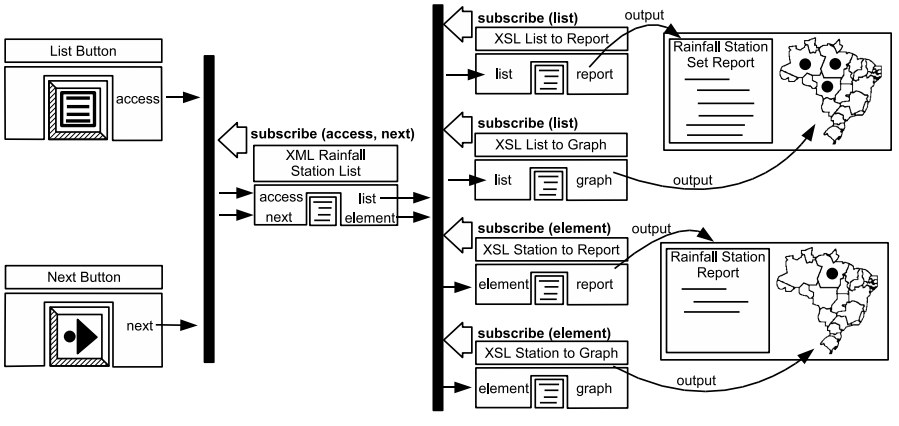


**Fig. 3.** Configuration that illustrates a content component composition

The configuration of components and connectors to form the application is stored in an XML document. This document contains the initial configuration of each component involved and the connectors configurations.

The representation of the connectors in the XML document depends on the architecture used to compose the components. In the publish/subscribe architecture, each connection is associated with a component subscription.

The framework of content components is designed to be flexible enough to adopt more than one architecture. Each kind of architecture is described and classified in a

taxonomy by an RDF description. An XML schema determines the format used in the XML file that stores the configuration. Based on the RDF description the framework determines the role played by the components and connectors and how the configuration file will be interpreted.

### 3.3 Component Storage and Retrieval Management

A digital content component is a combination of raw data, XML data and RDF data. The solution requires devising a database capable to interpret and manage each format, since XML and RDF data will be used for indexation and internal structure exploration purposes.

There are many possible solutions to this problem. They involve, among others, considerations and tradeoffs between constructing a native XML DBMS versus a relational DBMS that maps XML (and RDF) data to their structure and vice-versa. The data can be dynamically mapped from XML and RDF to a relational database and vice-versa. Shanmugasundaram [16] proposes a process to map XML to a relational database, which unifies many partial solutions to this problem. RDF, on the other hand, is based in a strong connected network of interdependent description elements. Therefore, besides the mapping, it is necessary to devise a model to retrieve coherent fragments of RDF data [3], preventing the transfer of large data volumes for each query.

Other possible solutions are the use of a native XML database, alone or combined with a relational database. On the one hand XML data can be stored and indexed in a native way, on the other hand there are limitations to current XML database implementations, such as the need for consistency mechanisms, transaction support, etc. To store RDF data in an XML database will require a costly process to code and decode the descriptions from XML data.

One of the purposes of this project will be to identify the best storage solution, as well to combine it with a procedure to index and retrieve components. It will result in a dynamic component repository, in the sense that components may be dynamically combined. Indexation will be based on adapting and combining three techniques to the context of content component: use of ontologies [20], component signature match [24] and behavior match [25].

In many cases the connection of two components will need other intermediate components, responsible for adaptations and conversions. The component repository manager can find such components comparing input/output interfaces; however, this process can produce many options for the same connection. Metadata associated with components can then be used to help choose the adequate component configuration, e.g., for a given quality requirement.

Another direction is to use the notion of DBMS monitoring to track the rate of use of each repository component, and the rate of adopted combinations. This will help guide the search for adequate component combination, based on previous experience, and can indicate the quality of component, based on its use rate.

### 3.4 Reference Implementation Framework

The project includes a construction of a reference implementation framework to: provide an infrastructure for components execution and intercommunication; interact with

a database to store/retrieve components; and provide support to component insertion, adaptation and reuse by applications. This framework will be based in the Anima infrastructure [14].

The Anima model is being extended to support any kind of content. This work includes the extension of Anima's software component model – presently based on modules coded in some program language – to deal with other types of process components, such as workflows and spreadsheets, which can act in many cases like software components. The construction of a workflow runtime component will be based in WOODSS. A spreadsheet runtime will explore a bridge to a spreadsheet system, such as the Universal Network Objects (UNO) [21], an interface-based component model of the OpenOffice system.

## 4 Concluding Remarks

This project combines work in Software Engineering with Semantic Web and database interoperability efforts. The main contribution is the formulation of an integrated view over these research areas, taking advantage from progress in the software components area to support development of applications in the Semantic Web. Another contribution is the four-part structure and the use of RDF to promote component specification interoperability, which will enhance component storage and indexation over different languages and standards.

The work proposed, under development, is based on previous experience in construction component-based applications for the educational domain [14], and on the use of scientific workflows, stored in databases, for reuse and interoperability of environmental applications [15].

**Acknowledgments**

This work was partially financed by UNIFACS, by grants from CNPq and FAPESP, and projects MCT-PRONEX SAI and CNPq WebMaps and AgroFlow.

## References

1. Anastassia Ailamaki, Yannis E. Ioannidis, and Miron Livny. Scientific Workflow Management by Database Management. In *Proc. 10th IEEE International Conf. on Scientific and Statistical Database Management*, pages 190–201, 1998.
2. Felix Bachman et al. Volume II: Technical Concepts of Component-Based Software Engineering, 2nd Edition. Technical Report CMU/SEI-2000-TR-008, Carnegie Mellon University, July 2000.
3. Alex Barnell. RDF Objects, November 2002. `http://www.hpl.hp.com/techreports/2002/HPL-2002-315.pdf`, accessed on 10/2003.
4. Wolfram Conen and Reinhold Klapsing. A Logical Interpretation of RDF. *Linköping Electronic Articles in Computer and Information Science*, 5(13), 2000. `http://www.ep.liu.se/ea/cis/2000/013/`.

5. Philip Dodds, editor. Sharable Content Object Reference Model (SCORM) – Version 1.2 – The SCORM Overview. Specification, Advanced Distributed Learning Initiative, October 2001. http://www.adlnet.org/screens/shares/dsp_displayfile.cfm?fileid=840, accessed on 10/2003.
6. Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.
7. Franca Garzotto, Luca Mainetti, and Paolo Paolini. Information Reuse in Hypermedia Applications. In *Proc. of the the 7th ACM conf. on Hypertext*, pages 93–104. ACM Press, March 1996.
8. IEEE L.T.S.C. Draft Standard for Learning Object Metadata – IEEE 1484.12.1-2002, July 2002. http://ltsc.ieee.org/doc/wg12/LOM_1484_12_1_v1_Final_Draft.pdf, accessed on 10/2003.
9. Robert Meersman and Amit P. Sheth. Special Section on Semantic Web and Data Management – Guest editor's introduction. *ACM SIGMOD Record*, 31(4):10–12, 2002.
10. Mikael Nilsson. IEEE Learning Object Metadata RDF binding, August 2002. http://kmr.nada.kth.se/el/ims/md-lomrdf.html, accessed on 11/2003.
11. Rubén Prieto-Díaz. Classification of reusable modules. In Ted J. Biggerstaff and Alan J. Perlis, editors, *Software reusability: vol. 1, concepts and models*, pages 99–123. ACM Press, 1989.
12. Rob Raskin. Semantic Web for Earth and Environmental Terminology (SWEET). In *Proc. of NASA Earth Science Technology Conference 2003*, 2003.
13. Ann Rockley. *Fundamental Concepts of Content Reuse*, chapter 2, pages 23–42. New Riders, 2000.
14. André Santanchè and Cesar Augusto Camillo Teixeira. Anima: Promoting Component Integration in the Web. In *Proc. of 7th Brasilian Symp. on Multimedia and Hypermedia Systems*, pages 261–268, October 2001.
15. Laura A. Seffino, Claudia Bauzer Medeiros, Jansle V. Rocha, and Bei Yi. WOODSS – A spatial decision support system based on workflows. *Decision Support Systems*, 27(1-2):105–123, November 1999.
16. Jayavel Shanmugasundaram et al. A general technique for querying XML documents using a relational database system. *ACM SIGMOD Record*, 30(3), September 2001.
17. Mary Shaw and Paul C. Clements. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. In *Proc. of the 21st International Computer Software and Applications Conference*, pages 6–13. IEEE Computer Society, 1997.
18. Michael K. Smith, Chris Welty, and Deborah L. McGuinness. OWL Web Ontology Language Guide – W3C Candidate Recommendation, August 2003. http://www.w3.org/TR/2003/CR-owl-guide-20030818/, accessed on 11/2003.
19. Colin Smythe, editor. IMS Content Packaging Information Model. Specification, IMS Global Learning Consortium, Inc., June 2003. http://www.imsglobal.org/content/packaging/, accessed on 11/2003.
20. Vijayan Sugumaran and Veda C. Storey. A Semantic-Based Approach to Component Retrieval. *SIGMIS Database*, 34(3):8–24, 2003.
21. Sun Microsystems. OpenOffice.org Developer's Guide, 2003. http://api.openoffice.org/docs/DevelopersGuide/DevelopersGuide.pdf, accessed on 01/2003.
22. Richard N. Taylor, Nenad Medvidovic, Kenneth M. Anderson, E. James Whitehead, Jr., Jason E. Robbins, Kari A. Nies, Peyman Oreizy, and Deborah L. Dubrow. A Component- and Message-Based Architectural Style for GUI Software. *IEEE Trans. Softw. Eng.*, 22(6):390–406, 1996.

23. Schawn Thropp and Mark McKell, editors. IMS Learning Resource Meta-Data XML Binding Specification, September 2001. `http://www.imsglobal.org/metadata/imsmdv1p2p1/`, accessed on 10/2003.
24. Amy Moormann Zaremski and Jeannette M. Wing. Signature Matching, a Tool for Using Software Libraries. *ACM Transactions on Software Engineering and Methodology*, April 1995.
25. Amy Moormann Zaremski and Jeannette M. Wing. Specification Matching of Software Components. In *Proc. of 3rd ACM SIGSOFT Symp. on the Foundations of Software Engineering*, October 1995.