CONSTRUCTION INFORMATION ACCESS THROUTH A "MALLEABLE FRAME"

Construction information access

Y. ZHU and R. ISSA
M.E. Rinker Sr., School of Building Construction, University of Florida, USA

Durability of Building Materials and Components 8. (1999) *Edited by M.A. Lacasse and D.J. Vanier*. Institute for Research in Construction, Ottawa ON, K1A 0R6, Canada, pp. 2246-2258.

Abstract

Information management is critical to the success of a construction project. Because of the fragmentation of construction information and the volatility of construction projects, information sharing and collaboration are important measures to achieve successful project information management. However, many current research projects focus only on shared project information. A gap between the shared project information and the non-shared project information still exists. This paper shows models and methodology to bridge the gap and make a document "malleable" according to user's needs by using contemporary computer technologies such as XML (eXtended Markup Language) and WDDX (Web Distributed Data eXchange). Some details about specifications for implementation are also presented.

Keywords: Shared project information, non-shared project information, document model, specification, hypermedia, XML, WDDX

1 Introduction

As the Internet based technology such as the WWW (World Wide Web), e-mail, e-commerce, e-bid and e-builder, becomes ubiquitous, many practitioners in the construction industry believe that "the Internet and its relatively simple working environment is the catalyst that will change the construction industry from paper to digital communication – particularly in the way of project collaboration and project management (ENR 1998)." Nevertheless, there is still a long way to go to achieve such a goal.

1.1 Problem description

Construction industry is notorious for information fragmentation (O'Brien 1993). From the object-oriented point of view, information, or attributes, related to an object is better to be placed together logically or be coherently and consistently connected through some mechanisms in order to facilitate information management (Martin and Odell 1998). However, since information related to the same object might be generated at different project stages by different people using different software tools in the construction industry, it is quite common that the information is physically, technically and semantically isolated. For example, a "slab-on-grade" object has information about its estimating, its scheduling, its budget, and its drawing. Because of the information fragmentation, the mapping of the relationships among those types of information requires manual translations. One solution to this problem is using a shared database, while the estimating, the scheduling, the budget and the drawings are different views to the same database (Fischer and Froese 1996). This solution works well for the shared project information, such as project drawings and manuals. However, for private information such as budget and cost accounts, even if the private information is modeled by using the same semantic model as that of the shared project information, there is still a need for bridging the private information and the shared information. This means because of the business constraints, the information fragmentation still exists even if the shared database approach is used.

Construction industry uses a lot of documents as communication media. One of the direct reflections of the above problems is that the information in the documents prepared at remote sites (by other project participants) may not be directly and automatically reused to retrieve related information by the local project information system. For example, a contractor may receive a Request for Change Order Proposal (RCOP) referring to a work item, "slab-on-grade", while the local system may have some associated information about that work item such as scheduling, budget, and estimating. The contractor normally need to know these types of information to prepare a return document, a Change Order Proposal (COP). However, when an architect is preparing the RCOP, he/she does not know what kind of information the contractor may need and where the information is. Therefore what the architect can do is to prepare a conventional type of document that only expresses what the document is and what kind of return information is expected. This approach is quite reasonably although not user friendly or user oriented.

Because of the existence such a problem, it relies on the document viewer to interpret the document and find the associated information. Very often what is in the document may not be what the user wants. Unfortunately the conventional document itself cannot do anything to help this situation. It will be a great step forward if there is a system that can not only present the document but also serve as a gateway to other related information. Such a system will definitely improve project participants' job efficiency.

This project is to use contemporary computer technology to build a prototype of such a system for the construction project participants, which is named **Kaleidoscope**.

1.2 Overview of Kaleidoscope

The purpose of **Kaleidoscope** is to provide project participants a system that allow them to automatically retrieve relevant information from local sources without resorting to other systems and also allow them to tailor the display according their own needs. **Kaleidoscope** is for Web-based applications and portable for different database systems and platforms.

Currently, there are three types of Web-based applications in construction industry. They are "fee-based project management service", "build it yourself solutions", and "Web-enabling software" (ENR 1998). The "fee-based project management service" such as Blue-Line/On-Line (1999) and Emerging Solutions (1999) provides a central server for project Web sites, allowing project participants to log on at different security levels to upload and download project information. Such systems are implemented as Web databases, providing users with organized data entry and retrieval mechanisms with different views. The "build it yourself solutions" such as the Aspects ProjectServer and Site-Builder by Black & Veatch (1999) focuses on similar technical approaches only without information outsourcing. The "Web-enabling software" such as the Primavera Project Planner (1999) can generate Web ready reports based on its own data sources. Nevertheless, those systems do not have a mechanism to deal with the problem mentioned previously. **Kaleidoscope**, as a complement to those systems, is designed to give project participants a more integrated working environment that provides them with the capability of dealing with information from different sources simultaneously.

1.3 Implementation technology

The implementation of **Kaleidoscope** relies on a mixed use of hypermedia design, XML (eXtended Markup Language), WDDX (Web Distributed Data eXchange), and Java.

Hypermedia design methods such as HDM (Hypermedia Design Method)(Garzotto et al. 1993), OOHDM (Object Oriented Hypermedia Design Method)[10] and RMM (Relationship Management Method)[11] are the major tools for designing specifications for dynamic hypermedia generation.

XML is a new Web technology. As a subset of SGML (Standard Generalized Markup Language), XML gives software engineers and developers a tool to automate the processing of Web documents (Bosak; St.Laurent; Microsoft Corp.; Sall; DOMS; XML; Walsh; Paoli and Workshop 1999). For **Kaleidoscope,** XML provides a fundamental meta-language for structuring construction documents and provides a tool for manipulating the documents as well.

WDDX is a new technology for integrating databases with different computing environments such as Java and JavaScript (Forta and Allaire 1999). The technology is based on XML by using XML syntax to form a system independent data format. In **Kaleidoscope**, WDDX is great helpful because **Kaleidoscope** relies on a neutral data format to transfer data between local databases and the **Kaleidoscope** environment, which is Java in this case.

The implementation of **Kaleidoscope** is done by Java program language.

2 System design

2.1 System architecture

Figure 1 shows the system architecture of **Kaleidoscope.**

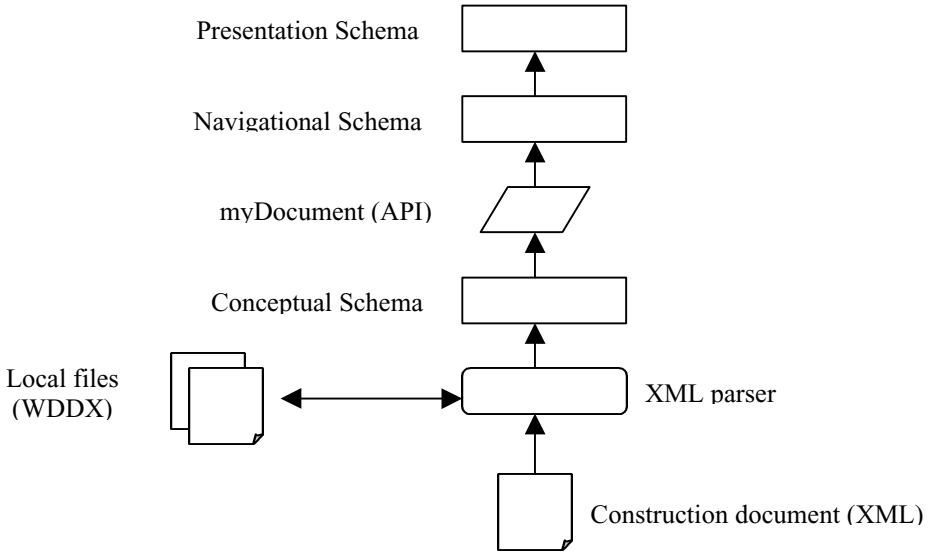

Fig.1: System architecture

The system starts with a construction document that is prepared in the XML format. The construction document is then parsed by a XML parser. In this project, the IBM parser is used. During the parsing process, some units may need to get local information. This is done by sending out a universal key and other information to local systems in the WDDX format. The results from local systems, if any, are stored in WDDX format as well and sent back to the parser. The results after parse are several trees, namely a tree structure representing the construction document and some tree structures representing local information. The local information is always related to some different document units. Therefore, logically there is only one tree. The tree forms the conceptual schema. To access to and manipulate the tree, myDocument, APIs (Application Programming Interfaces), is provided. Although we can always use W3C DOM (World Wide Web Consortium Document Object Model) APIs, myDocument provides a simplified set of APIs that are designed for applications involving local information. The navigational schema defines access structures. **Kaleidoscope** provides three kinds of access structures, which are the document index, the local information type index and the local information mixed index. The implementation of these access structures and the related presentations rely on the services of myDocument. The presentation schema organizes presentations according to tasks and provides interface features for user interactions.

2.2 Conceptual schema design

The application domain of **Kaleidoscope** is relatively simple yet dynamic. It is simple because only a few entities may actually be involved. Those entities only include a construction document and related local information. Therefore it is not like

many other hypermedia applications that may have tens or hundreds of entities. However, it is more dynamic because for different types of construction documents, the information types in the document as well as related local information types may change. This means the hyperbase of **Kaleidoscope** is ever changing according to different construction documents. For most of other hypermedia projects, they always have very clear and stable hyperbase in terms of entities and their attribute types. To solve the problem, **Kaleidoscope** uses a more generic conceptual schema.

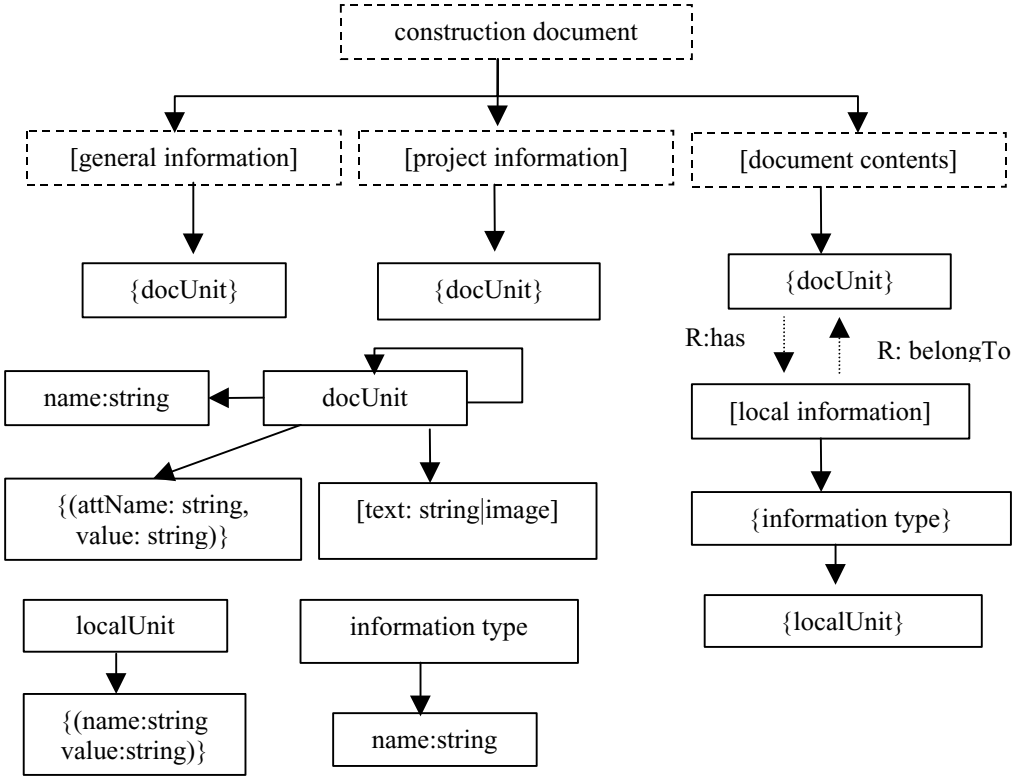

Fig. 2: A "conceptual schema" for Kaleidoscope

Following the principles and the concepts of HDM, **Kaleidoscope** structures construction documents into three levels, namely entities, components and units. The documents themselves represent entities. Each document contains three types of components, general information, project information and document contents. General information includes units such as document titles, document dates, document id, document senders, document receivers, document checklists and signatures. Project information includes project names, project ids and project participants. Document contents include names of work items, descriptions and attachments. Units of the components are equally referred to as "docUnits".

Similarly, information retrieved from local databases is structured in the same way. On the top, local nodes stand for entities, below which local information types, such as scheduling and estimating stand for components and their attributes are the units. The units here are referred to as "localUnits" to be distinct from the "docUnits".

In order to cover different construction documents and different local information, the conceptual schema is designed as shown in Figure 2 and Figure 3 respectively.

The notations used in Figure 2 and later are only to serve the purpose of this project. Explanations of the notations are as following:

- The boxes in dash lines contains structure information only, and do not have any document information. The boxes in solid lines represent both the structure and the contents of a document.
- "[ ]" means that the elements inside it may appear zero or once only; "{ }" means that the elements inside it may be generated zero or more times. "( )" means that the elements inside it appear in pair; Elements that have no braces or brackets around them, such as the "name:string" of "information type", must appear exactly once.
- "R" stands for special relationship, represented by dash arrowhead lines. Therefore here we have two special relationships. A "docUnit" may "has" a "localUnit" and thus the "localUnit" "belongTo" the "docUnit". Other relationships, such as the structural relationship, the perspective relationship and the application relationship are equally represented by solid arrowhead lines.
- "Text" and "image" stand for the type of the elements. " | " means "or". And ":" is for type explanation.

Local information is stored in a neutral data format shown as Figure 3.

```
<localNode>
<id>id</id>
<uid>uid</uid>
{<localInformatioType {name="value"}>}
</localNode>
```

Fig. 3: Data format for local information

Figure 3 shows that one local node, which corresponds to "local information" in Figure 2, may have zero or more types of local information. Each type of local information may have zero or more pairs of name and value, which correspond to the name and value pairs of the "localUnit" in Figure 2.

In this way, local information that is related to one "docuUnit" may be grouped together under one node. For example, the "slab-on-grade" mentioned earlier may have local information of scheduling and budget. Under the same node, "localNode", scheduling and budget are two local information types. Within each type, such as scheduling, name and value pairs, such as "early start = 1/12/98", are included.

2.3 Navigational schema design

The navigational design defines three access structures for users to manipulate the information defined in the conceptual schema. These structures are a document index, a local information type index and a local information mixed index. These indexes are defined in Figure 4.

In the navigational schema shown in Figure 4, there are several new notations used. They are:

- "<!-- … -- >" stands for comments.
- "docUnit.Name" means the name of this "docUnit". "." is used for representing this relationship.
- "confirm" stands for a confirm bit that captures user's interactions. The "confirm" is not a part of document information, therefore it is not shown in the conceptual schema. However, logically each "docUnit' and each "localUnit" have a confirm bit.

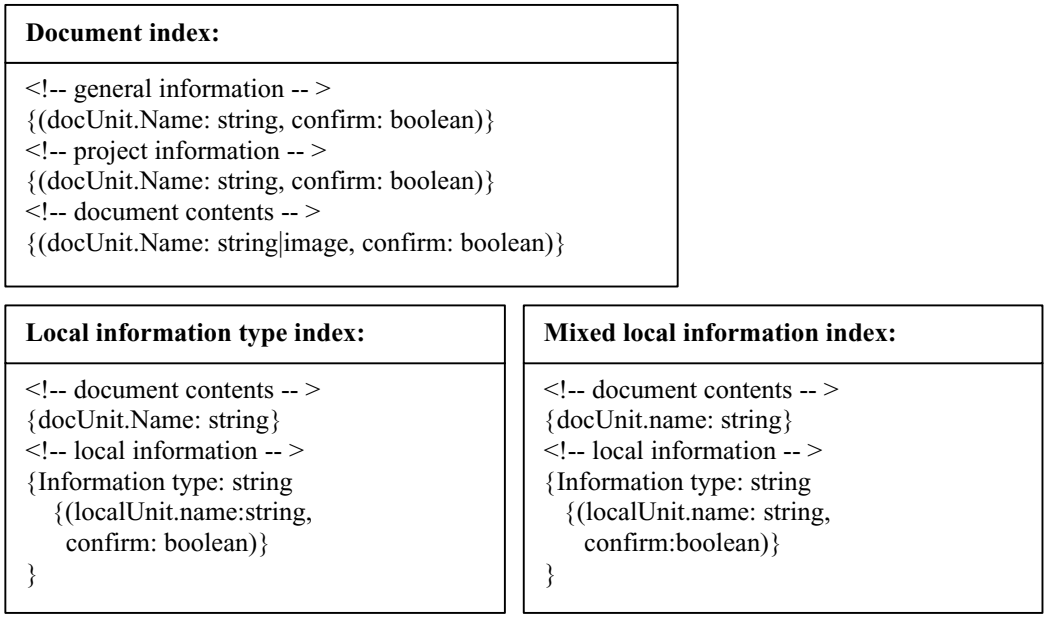

Fig. 4: Navigational schema of Kaleidoscope

The local information type index and the local information mixed index are always associated with a certain document unit. They cannot exist alone.

The index structures control the traverse across the hyperbase developed in the conceptual schema. They also provide access points. The actual display is determined according to user's interactions and the "confirm" bits will indicate to the system what to display.

Corresponding to each of the above indexes, there are three types of display templates. The specifications of these templates are shown in Figure 5.

In the template for document display, the only action is to display the "text" of a "docUnit" if the "confirm" of the "docUnit" is true. The condition is wrapped inside a

"{}", which means that there may be more than one "docUnit" selected. Thus the "display()" is wrapped inside a "{}" as well to be congruous with the condition.

The templates for the local information type display and the local information mixed display are similar. Both of them have a "isTrue(docUnit.R:has)" condition. This condition is to check if a "docUnit" has a local node. If a "localUnit.confirm" is "true", the "name" and "value" of this "localUnit" are displayed. In order to make the display clear, the "information type" of this "localUnit" and the "name" and the "text", if any, of the "docuUnit" that the "localUnit" indirectly belongs to are displayed as well.

The differences between the two local information index templates are that the mixed local information display allows simultaneously display of different information types, thus it needs to check zero or more "confirm" bits. The other template however only needs to check zero or once. Consequently, the display of the mixed local information is wrapped inside a "{}", while the display of the other template does not.

Construction Document Display:

Condition: {docUnit.confirm = =true}
Action: {display(docUnit.text)}

Local Information Type Index Display:

Condition: isTrue(docUnit.R:has) = =true
[localUnit.confirm = =true]

<!-- document unit -- >
Action: display(docUnit.name,
docUnit.text)
<! -- related local information -- >
Action: display(information type)
display(localUnit.name,
localUnit.value)

Mixed Local Information Index Display:

Condition:
isTrue(docUnit.R:has) = = true
{localUnit.confirm = = true}

<!-- document unit -- >
Action:
display(docUnit.name, docUnit.text)
<! -- related local information -- >
Action:
{display(information type)
{display(localUnit.name
localUnit.value)}
}

Fig. 5: Specifications for various display templates

2.4 Presentation schema design

The presentation schema design includes presentation design, user interface design and process design. The purpose of the presentation schema is to organize presentations for different states of different tasks.

2.4.1 Presentation design

The three display templates discussed in the previously section are abstract designs, they do not say how the information is presented to a user. Presentation design includes specifications for such purposes. Again, the document layouts are subject to document types. Thus, here we only consider some generic feature, while leaving the actual document layouts to different applications.

The rules for presentation are based on matching. For instance, the document index shown in Figure 4 determines what is to display by set the “confirm” to “true” and the display templates shown in Figure 5 tells what to do under certain conditions. Now the presentation template shown as Figure 6 determines how the information is perceived by users. The presentation templates inherit conditions and display functions from the display templates in Figure 5. They have additional conditions however showing the rule for generating presentations. The rule is matching. This is done by the condition, “This.name = = docUnit.name” or “This.name = = localUnit.name”. Here, “This” refers to the unit selected by the user in the index template. Once “This” unit is selected, the name of “This” unit is send back for searching a unit with the same name in the document tree and the local information three. If the name is found, and the corresponding content, “text” or “name”, exits, the corresponding display is carried out and presentation is configured by “set ( )” .

The presentation templates are generic templates for various kinds of units.

Document Unit Presentation:	**Local Unit Presentation:**
Condition: inherit conditions from the display template; This.name= =docUnit.name; isExist(docUnit.text)= =true; Action: inherit display from the display template set(font, fontColor, alignment,..)	Condition: inherit conditions from the display template; This.name= =localUnit.name; isExist(localUnit.name)= =true; Action: inherit display from the display template set(font, fontColor, alignment,..)

Fig. 6: Unit presentation specification

2.4.2 User interface design

Although the focus of this project is not on interface features, the interface design is an unavoidable part. Thus here we only work on the bottom line to show that the designed mechanism works. Following we will discuss main user interface features of **Kaleidoscope**.

Major screen layouts include a toolbar for different menus and a main presentation window. The menus include “Document” to set views for documents, “Local Information” to set views on one type of local information, “Mixed Local Information” to set views on multi-types of local information and “Process” to switch states. Obvious, “Document” is the implementation of the document index, and “Local Information” is the implementation for the local information type index and “Mixed Local Information is the implementation for the local information mixed index. Buttons will be used as triggers for those menus. The contents of these menus are generated dynamically according to different documents.

Once a document is opened, the document index is always active to users to set default views or tailor a display. Since the local information type index and the local information mixed index are always related to a certain unit in the document, these indexes are not active until the user selects a proper unit. To help users identify what units have local information, the system will attach a checkbox to the right unit and

also make that unit appear different from others. Therefore, if there are many units that have local information, **Kaleidoscope** can provide local access to one unit a time.

A construction document may include some attachments. They are shown in separate windows. Once a document is opened, the attachments are shown automatically by default unless the user reset it in the "Document" menu.

3 Conclusion

This paper presents models, specifications and methodology for developing a hypermedia system that will make a project document prepared by others malleable to the document viewer. This gives project participants some capability manipulate the document in the way they want. This also break the limit of conventional documents that the views of those documents are set and fixed by the party who prepares the document. The document in **Kaleidoscope** is thus more user friendly.

However, **Kaleidoscope** is based on two major assumptions that require further studies.

3.1 Assumption one

This project is to use XML to represent structured construction documents. Currently construction documents are not structured in a way that fits the requirements of XML. Most importantly, the vocabulary set of construction documents needs to be assumed. For instance, this project may use following XML tags denoting some nodes in a construction document, shown as Figure 7.

```
<documentType docName="changeRequest" id="CH-123">
Request for Change Order Proposal
</documentType>
```

Fig. 7: An example of a construction document in XML format

In the above example, the node name, "documentType" and the attribute name, "docName", are part of construction document vocabulary. This vocabulary has to be an industrial standard, or de facto standard. Currently, such vocabulary does not exist yet, let alone to say an industrial standard.

3.2 Assumption two

One of the major features of **Kaleidoscope** is its ability to combine a document with some related local information. This project however can only show a theoretic solution, because one key technical problem has not been solved in building construction. This technical problem is of a universal index (uid) system. Once a node is designed to search for local information, the node passes its uid to the local system so that the local system can do search and retrieve based on this uid. In the building construction, such an index does not exist currently. What we have now is a very rough uid, such as the CSI (Construction Specification Institute) indexes. Such indexes cannot meet the requirement of this project. Therefore in this project it has to

be assumed that the work items in a construction document and their associated information in the local system can be uniquely identified.

4 References

Information Technology, the Internet and You (1998) ENR (Engineering News Record), June 1, pp c3-c33.

O'Brien, M. (1993) Electronic Data Interchange and the Structure of the UK Construction Industry, Construction Management & Economics, v11, n6, Nov., pp443-453.

Martin, J and Odell, J. (1998) Object-oriented methods : a foundation Upper Saddle River, NJ : Prentice Hall.

Fischer, M. and Froese, T. (1996) Examples and Characteristics of Shared Project Models. Journal of Computing in Civil Engineering Vol 10 No. 3, pp174-182.

Blue-line / on line Web, http://www.bluelineonline.com/

Emerging Solution, http://www.emergingsolutions.com/

Black & Veatch Web, http://www.bv.com/

Primavera System Inc., http://www.primavera.com/

Garzotto, F., Paolini, P. and Schwabe, D. (1993) HDM---A Model-Based Approach to Hypertext Application Design, ACM Transactions of Information Systems, Vol. 11, No. 1, pp1-26.

Isakowitz, T., Stohr, E.A. and Balasubramanian, P.,(1995) RMM: A Methodology for Structured Hypermedia Design, Communication of the ACM, Vol. 38, No. 8, pp34-44.

Schwabe, D., Rossi, G. and Barbosa, S.D.J. (1996) Systematic Hypermedia Application Design with OOHDM, Proceedings of the Seventh ACM Conference on Hypertext, pp116-128.

Bosak, J., XML, Java, and the Future of the Web, http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm.

St.Laurent, S., XML Essays. http://members.aol.com/simonstl/xml/

Microsoft Corporation, XML White Paper, http://www.geocitites.com/WallStreet/Floor/5815/xmlwhitepaper.htm

Sall, K., XML: Structuring Data for the Web: An Introduction. http://wdvl.internet.com/Authoring/Language/XML/Intro/

Document Object Model Specification, http://www.w3c.org/TR/REC-DOM-Level-1/expanded-toc.html

XML Specification, http://www.w3.org/TR/PR-xml-971208

Walsh, N., A Technical Introduction to XML. http://www.arbortext.com/nwalsh.html

Paoli, J., et al, Building XML Parsers for Microsoft's IE4. http://www.oasis-open.org/cover/paoliXMLOreilly.html

Workshop on XML. http://www.microsoft.com/xml/contents.htm

Forta, B., Using WDDX to Create Distributed Applications, http://www.allaire.com

Allaire, J., The Emerging Distributed Web, http://www.allaire.com