5

XML Software Infrastructure

Executive Summary

The XML paradigm naturally supports robust components

The previous chapters examined the theory behind the XML paradigm. But realizing practical benefits requires successfully deploying XML-based systems. As with any software technology, the ability to leverage off-the-shelf infrastructure greatly improves your chances of successful deployment. Luckily, the XML paradigm's high degree of standardization and openness has resulted in rich layers of software infrastructure for both data-oriented and content-oriented applications.

You must assemble a strong foundation of core infrastructure

This chapter focuses on the basic infrastructure used in many XML projects. This core functionality supports the rest of the application. Of course, not every application needs every piece of infrastructure, and there are often choices within an infrastructure category. Understanding when different components are necessary and the issues in choosing a particular component will help you ensure that your projects have a strong foundation for success. Figure 5-1 presents a conceptual model of the basic infrastructure.

Fundamental components perform basic XML operations

As you can see, fundamental components provide the basic services for the rest of the categories. Fundamental components, such as XML processors that implement the XML specification and XSLT processors that implement the XSLT specification, are already widely available from open source projects and software vendors. They have had plenty of time to mature and provide a robust infrastructure foundation.

Choosing a storage solution depends on the type of application

Because XML is about data and content, these fundamental components implicitly assume they have a means of persistent storage. A simple application may use the local filesystem, but one that is more sophisticated requires mechanisms with greater reliability,

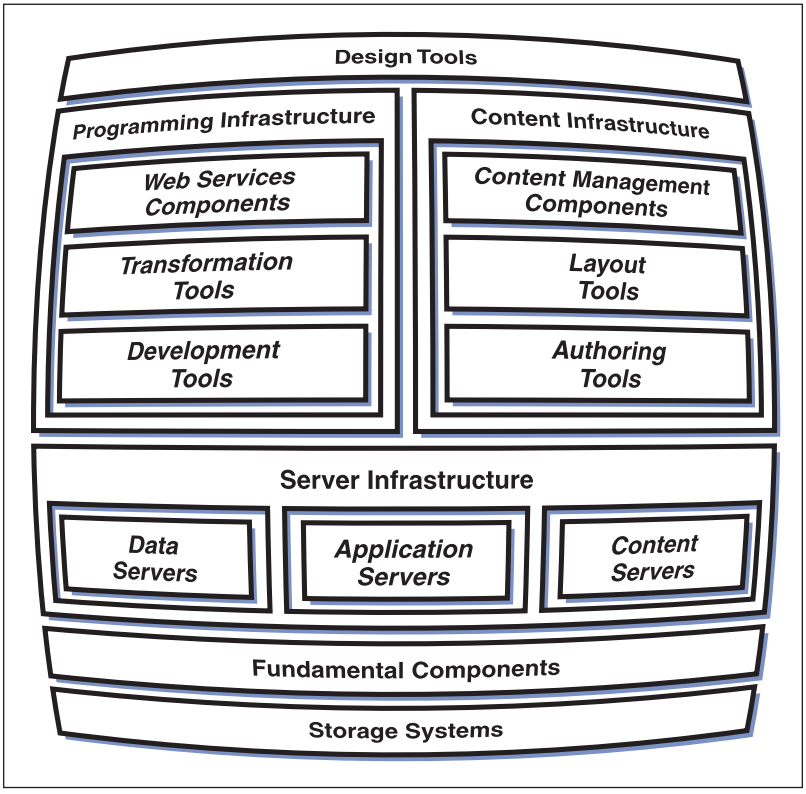

Figure 5-1: Conceptual Model of XML Software Infrastructure

scalability, and flexibility. There is no single optimal storage solution. The appropriate choice of database management system, content management system, or native XML store depends on the intended purpose and access pattern of your information.

Moving and processing XML requires server infrastructure

Enterprise and Web architectures depend on various types of servers to manage data, execute behavior, and distribute content. For developers and authors to move XML documents around in this environment, servers must be at least XML-aware and may need XML-specific functions. Data servers must provide interfaces for

accessing data in an XML format. Application servers must deliver scaleable execution of fundamental components. Content servers must facilitate the delivery, distribution, and cataloging of XML-packaged content.

One set of components helps process XML as data

Above the server layer, infrastructure clearly diverges along two paths: the data path and the content path. The data path includes XML components related to the generation, processing, and movement of machine-readable XML data. Development tools enable programmers to rapidly create software that moves data between internal data structures and XML documents. Transformation tools enable programmers to convert data between XML formats or between XML formats and other data sources. Web services components help programmers package, process, and interpret XML-encoded messages.

Another set of components helps process XML as content

The content path includes components releated to the authoring, presentation, and distribution of human-readable XML content. Authoring tools facilitate the creation and editing of XML documents. Layout tools facilitate the presentation of XML documents in a variety of media. Content management components facilitate collaboration, packaging, and delivery of XML documents.

Design tools drive the interpretation of XML by the other components

The two paths of XML infrastructure merge again with design tools. The process of information design is what sets XML apart from other software technologies. This process provides the programmer or author the means to design formats for XML, whether as data or content. In fact, it is the information design process that determines whether your XML documents are data or as content. While this process should actually occur very early in application development, the results drive the choice of other components, which is why design tools appear as the top layer in Figure 5-1.

Fundamental Components

All XML applications rely on fundamental components

XML defines a standard for encoding documents to facilitate seamless information exchange. Obviously, for such exchanges to actually take place, all parties must be able to process XML documents. If they use related feature standards, such as XSLT or Schema, they must be able to handle those formats. Applications need these basic processing capabilities to realize the benefits of the XML paradigm. The software engines that perform this essential work are the fundamental components.

XML processors are the most important fundamental component

The biggest workhorse in the XML paradigm is, of course, the XML processor. This piece of software can read an XML-formatted file and provide the encoded data to other software components. Most XML processors also understand DTDs, Schema, and Namespaces. Many can process XPath statements. Other fundamental components, such as those for processing XSLT and XLink, rely on the XML processor because the standards they implement use XML syntax. Therefore, a basic understanding of XML processors is essential to understanding the architecture of XML-based applications.

There are tree-based and event-based XML processors

An XML processor is typically embedded in an application. It reads the physical files associated with a document and converts the document text into programming constructs accessible to the application logic. There are two basic types of processors: tree-based and event-based.

Tree-based processors create a hierarchical data structure

Figure 5-2 shows the operation of a tree-based processor. It accepts an XML document as input and then parses the document to create a hierarchical data structure that is an in-memory representation of the data contained in the document. Optionally, it may also accept an XML DTD or Schema, in which case it verifies that the document follows the specified rules.

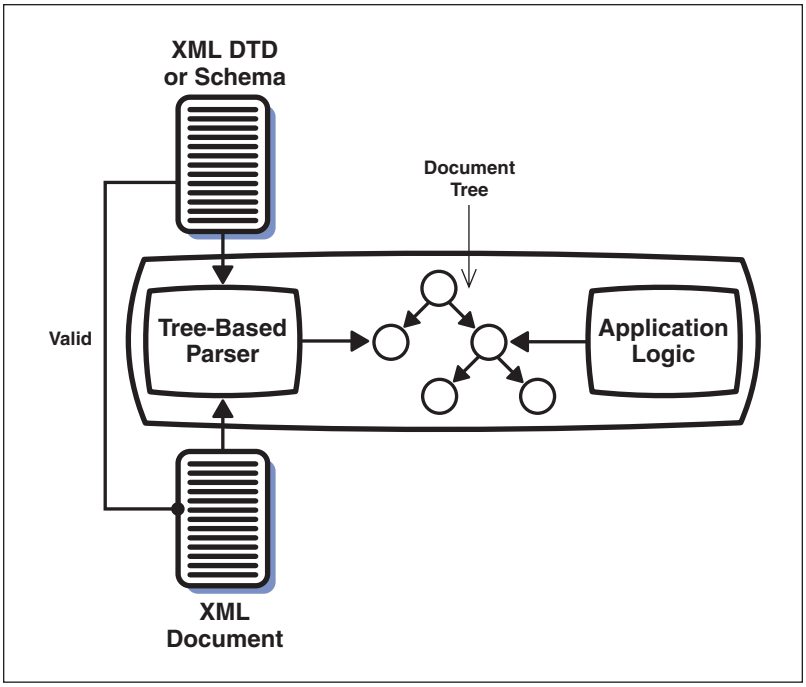

Figure 5-2: Tree-based XML Processor

Developers access the document tree through an API

Application logic accesses the hierarchical data structure through an **application programming interface (API)** provided by the processor. The **Document Object Model (DOM)** is another W3C Recommendation that defines the characteristics of the document tree and the API for manipulating it. A tree-based processor does not have to support DOM to support XML; it may have a proprietary type of data tree and API. However, DOM support is a good idea because it ensures that developers can learn this one API and use any compliant processor. DOM and proprietary APIs allow developers to write application logic that moves through the document's tree of data, extracting and evaluating the information required to execute application functions. In addition to DOM or other tree API, many tree-based processors also support the use of

XPath expression to access nodes within the data hierarchy as described in Chapter 3.

An application can use a tree-based processor to create documents

This process can also work in reverse. An application can use the processor's API to create a new tree, add nodes to the tree, and fill them with data. The processor uses this newly created tree of data to create a corresponding XML document. In this case, the application logic and processor work together as a document generator. Optionally, it can check to make sure that the generated document conforms to a specified DTD or Schema.

Event-based processors require less memory

Tree-based processors are highly effective for applications that need random access to document elements. The application can use the API to navigate the path to any piece of data. However, building a complete tree for every document can consume a substantial amount of memory. Event-based processors avoid this cost because they do not create an in-memory data structure for the entire document.

Applications must explicitly request information about elements

Instead of creating a data structure and letting the application logic access it, event-based processors send just event descriptions to the application logic. These events include the beginning and end of each element but not its attributes or text content. If the application logic is interested in that particular element, it requests additional data from the processor. This approach is highly effective when an application accesses each element sequentially or only needs access to a known subset of elements. Figure 5-3 shows how this process works. As with tree-based processors, event-based processors can also take an optional XML DTD or Schema against which it validates the input document.

SAX events include only the start and end of elements

The **Simple API for XML (SAX)** is a commonly used event API for XML processors. Example 5-1 illustrates the types of events a SAX-based processor would generate when processing the order

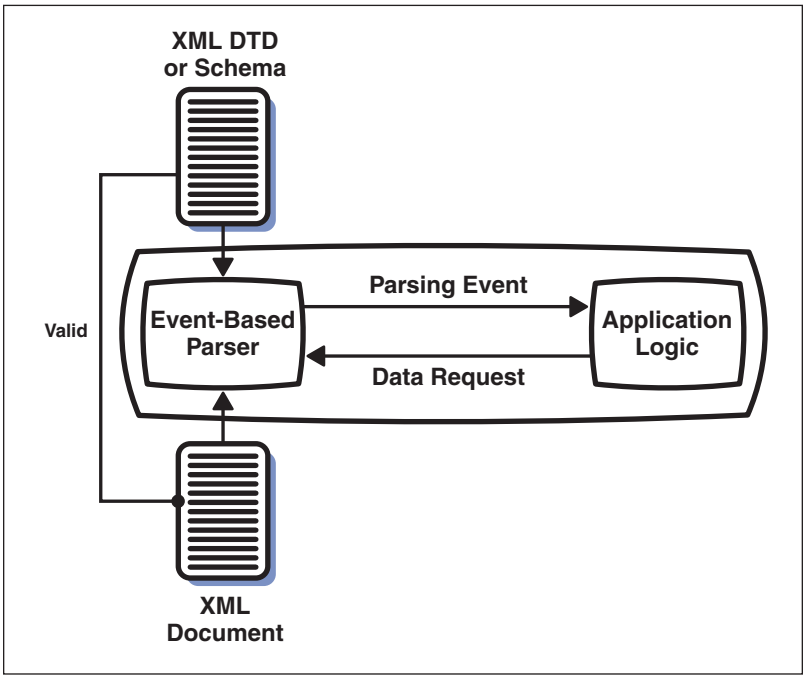

Figure 5-3: Event-based XML Processor

document in Example 2-6. As you can see, the processor does not include any element content or attribute information. When the application logic receives an event signifying the beginning of an element whose content it wants to access, it must specifically request information about that element.

Example 5-1

```
Start document
  Start element: Addresses
    Start element: Address
      Start element: FirstName
      End element: FirstName
      Start element: LastName
      End element: LastName
```

```
    ...
  End element: Address
  Start element: Address
    ...
  End element: Address
End element: Addresses
Start element: LineItems
  Start element: LineItem
    Start element: Product
    End element: Product
    Start element: Quantity
    End element: Quantity
    ...
  End element: LineItem
  Start element: LineItem
    ...
  End element: LineItem
End element: LineItems
...
End Document
```

Many processors support both tree- and event-based modes

In practice, many XML processors support both tree-based and event-based APIs. This support stems from the fact that it's easy to build a tree-based processor on top of an event-based processor. To build the in-memory data tree, the tree-based processor acts like any other application from the perspective of the event-based processor. The only difference is that it always requests all the information about an element. It then takes this information and puts it in the data tree. Figure 5-4 shows how this process works. Developers that want a tree-based API enable the tree assembly functions of the processor, while developers that want an event-based API suppress them. The advantage of this dual architecture is that both types of developers can use the same component.

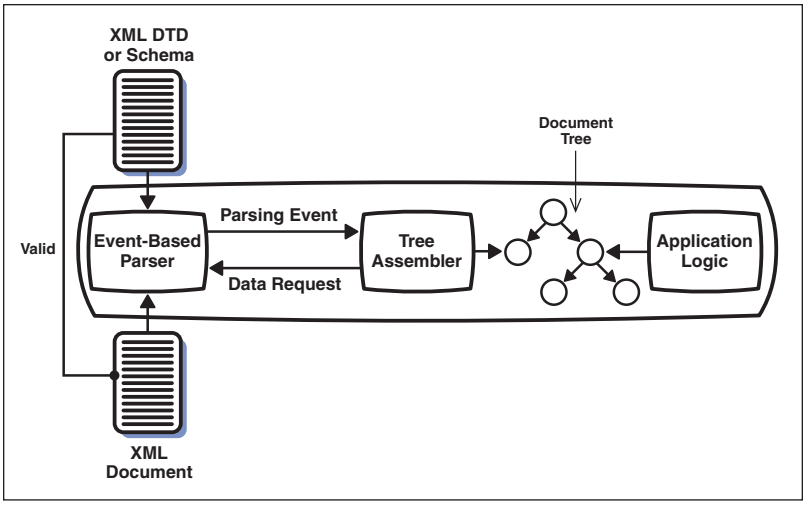

Figure 5-4: Dual Mode XMl Parser

Most procedssors can handle well-formed and valid documents

Figures 5-2, 5-3, and 5-4 show processors accepting a DTD or Schema as input along with the XML document. In such cases, the processor validates the document against the rules specified in the DTD or Schema. Such processors are called **validating processors**. Most of the major validating processors now support Schema as well as DTDs. However, applications do not have to use the validating features of such processors. There are also nonvalidating processors designed for applications that require only well-formed documents.

Open source and commercial processors are widely available

Both open source projects and software vendors provide XML processors. The Apache Software Foundation provides the popular Xerces open source processor for both Java and C++, although the Java version is usually somewhat ahead of the C++ version in terms of features. Microsoft provides an XML processor as part of its operating systems and makes this component easily accessible from its development tools. Oracle has C, C++, and PL/SQL processors for use with its database and middleware products. There are open

source processors for many other languages, including JavaScript, Perl, and Python.

There are also a wide variety of XSLT processors

After XML processors, the next most important fundamental component is the XSLT processor. XSLT has rapidly achieved widespread acceptance and is an important technology for many data- and content-oriented applications. The Apache Software Foundation provides an XSLT processor called Xalan in both Java and C++ flavors. Microsoft and Oracle also provide XSLT companions to their XML processors. There are other less known open source and commercial XSLT processors, but the complexity of XSLT has limited their proliferation.

XSLFO and XLink processors are only now becoming available

The availability of XSLFO formatters and XLink processors is not as great. These feature standards are just beginning to see substantial use, and their areas of application lend themselves less well to distribution as components. XSLFO is relevant primarily as part of publishing systems, while XLink is primarily relevant as part of hypertext systems. This is not to say that corresponding fundamental components are not available, only that their number and quality are less than for XML and XSLT. At the time of this writing, there were a few open source XSLFO formatters, notably FOP from the Apache Software Foundation and PassiveTeX from the Text Encoding Initiative. Commercially, Antenna House's XSL Formatter and RenderX's XEP Rendering Engine were available. All tend to focus on PDF output. For XLink, empolis' X2X and Fujitsu XLink Processor (XLiP) both handle extended links but require fees for commercial use.

Storage Systems

There are three different choices for storing XML

Once you have acquired the necessary fundamental components, you can create, access, and manipulate data in XML documents. But as with all data, you usually want some form of persistent

storage that's more robust than the local filesystem. It is not uncommon for projects using XML to stall while figuring out how to address the storage issue. The confusion stems from the fact that there are three vastly different choices: a database management system (DBMS), a content management system (CMS), or a native XML store. The appropriate choice depends on the characteristics of your XML data.

In data interchange, XML is an intermediate representation

What if you use XML as a data interchange format? In this case, a source application encodes data from its own native format as XML and a target application decodes the XML data into its own native format. XML is an intermediate data representation. Both the source and target applications already have persistent storage mechanisms, almost certainly DBMSs of one sort or another. There is really no need to persistently store the XML documents themselves, except perhaps for logging purposes.

The point of data interchange is to put data in a DBMS

In fact, the entire purpose of the interchange format is to combine data from an external source with the rest of the data in the DBMS. If you want to access or search data from these interchange documents along with data already in the DBMS, you need to convert it from XML to the DBMS's native format. You may take this approach even further by making XML the lingua franca among different data sources. The discussion of Data Servers in the Server Infrastructure section below addresses this option. But even in this sophisticated case, XML remains an intermediate data format. The data is ultimately translated and stored in an existing DBMS.

In content production, XML storage is only part of the problem

What if you use XML as a content format? In this case, authoring tools generate content as XML, and layout tools generate stylesheets for displaying this content. But content production usually requires higher-level features beyond storage, such as collaborative authoring, rendering to different media, and indexing documents. The

following discussion of Content Provisioning Components in the Content Components section below examines the need for these higher-level features in more detail. Moreover, you may also have content in other formats that you must manage alongside the XML documents.

Content management systems (CMSs) provide a complete solution

In this case, you probably want to use a CMS that addresses persistent storage in conjunction with these other needs. Because most commercial CMSs evolved with the use of SGML, vendors have found it fairly easy to add excellent XML support. So if XML is a content and layout representation, use a CMS. CMS products with XML capabilities include BroadVision Publishing Center, Chrystal's Astoria, Documentum4i, Interwoven TeamSite, OmniMark Technologies' OmniMark, Red Bridge Interactive's DynaBase, SiberLogic's SiberSafe, and Vignette's Content Suite. The discussion of Content Management Components in the Content-Related Components section that follows covers the features of CMSs in more depth.

Business document and orchestration applications use XML operationally

What if you use XML as an operational data format? Operational data is data that directly drives an application or process. Usually, DBMSs maintain operational data, but there are two cases where XML is likely to be the format. In the first case, XML is the format for an important work product of some kind. As discussed in Chapter 4, the business document architecture uses XML in precisely this manner. Each document represents a completed work product exchanged between organizations. Certainly, organizations will break down this document and map certain portions to corresponding DBMSs. However, the XML document is the starting point for driving this downstream processing and the ultimate point of reference for auditing. In the second case, XML is the format for instructions used in executing a process. Chapter 4 also discussed the emerging class of orchestration applications that use an XML

format to describe the assembly of software components or the workflow for business processes.

Native XML stores work best for operational XML data

In either of these situations, neither a traditional DBMS or a CMS is appropriate. You need to use the XML document as a single unit but still index its internal contents. Traditional DBMSs do this poorly because they either have to disassemble the document into their internal formats or create special functions for treating documents as **Large Objects** (LOBs). Traditional CMSs do this poorly because they are not optimized for subsecond response under high request loads. So if XML is an operational data representation, use a native XML store. Such products include Ipedo XML Database, IXIASOFT's TEXTML Server, NeoCore XMS, Software AG's Tamino, and XYZFind Server. The products mentioned in the Data Server discussion of the Server Infrastructure section below can also store native XML data and are particularly useful when your application has a combination of native XML and traditional DBMS data.

Server Infrastructure

XML server support is important in Web architectures

With an appropriate place to persistently store XML data, the next concern is distributing and manipulating this data. In modern Web application architectures, servers play critical roles in assembling, processing, and distributing information. Adding XML support to your server infrastructure mostly involves making sure that existing servers are XML-enabled, with perhaps the installation of a few XML-specific components. Most importantly, you must verify that XML capabilities meet the scalability and reliability demands of all server functions.

There are data, application, and content servers

In general, there are three types of server components: data servers, application servers, and content servers. Data servers access, aggregate, and format data. Application servers execute business logic

components and mediate distributed business processing. Content servers facilitate the acquisition of content, enhance its accessibility to users, and apply formatting. Figure 5-5 shows these different types of servers working together in a typical web application environment. This type of server web provides the conduit for propagating XML documents within an enterprise and throughout the Internet.

Products may combine server types in different ways

While Figure 5-5 shows each server component as a distinct node, this arrangement isn't necessary. Server software may combine these components in different ways, and in fact, different combina-

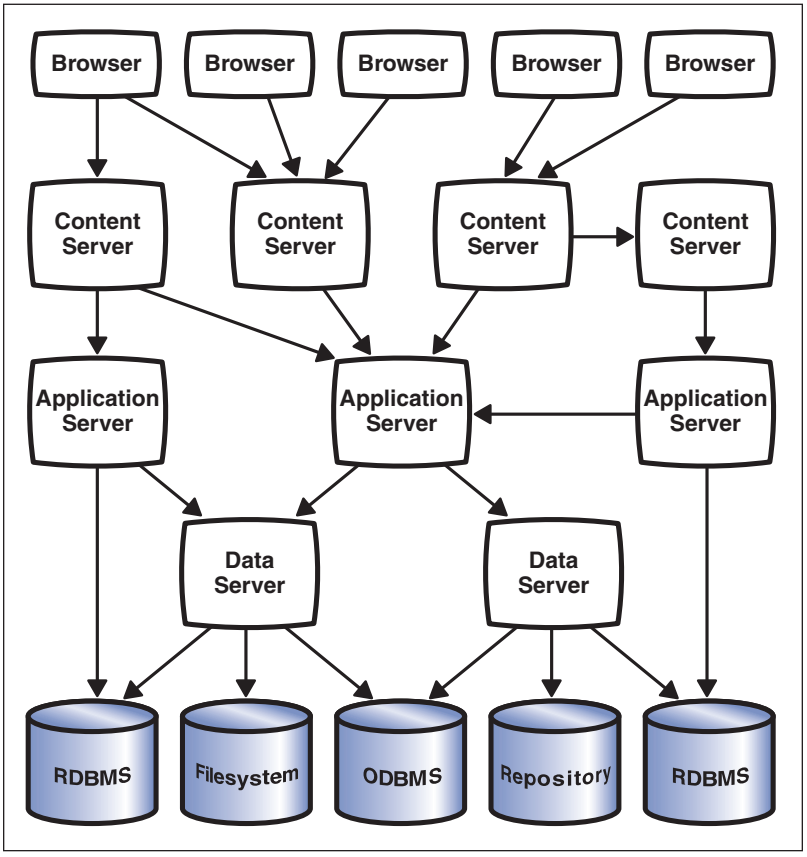

Figure 5-5: Types of Server Components

tions lead to distinct product segments. Integration servers combine data server functions to aggregate information from multiple sources with application server functions to control the flow of business processes. Portal servers combine data server functions to access information from multiple sources with content server functions to filter this information based on user requirements. Personalization servers combine application server functions to calculate user needs with content server functions to dynamically customize their experiences. By understanding the roles of the three types of server functionality, you can evaluate whether such a combination suits your needs.

Data Servers

Sophisticated architectures may strain DBMSs

DBMSs inherently constrain the use of data. They have to choose a particular paradigm, such as relational or object. Relational DBMSs with normalized tables optimize the combination data in different ways. Object DBMSs with associated instances optimize the traversal of information webs. Within a given paradigm, each individual database has a particular structure limiting the types of information it can store and the access patterns it supports. DBMSs do a wonderful job of managing data when a given database must support only a few types of applications and when each application relies on only a few databases. However, when a given database must support a wide variety of application types or a given application must rely on many different databases, satisfying these demands often tax DBMSs to their limits. In such cases, an XML-enabled data server can improve flexibility and performance.

Data servers facilitate a web of many DBMSs and many applications

XML broadens the use of data. The ability to quickly design special purpose data formats encourages the combination of information managed in different databases. So while data servers have existed for some time, XML's emergence as a solution to information exchange problems has elevated their role. Data servers perform three

major functions: (1) they unify the data access interface to simplify application development, (2) they aggregate data from different sources to deliver customized packages of information, and (3) they consolidate requests to DBMSs to improve performance. XML requires special support only in the first two functions. Because optimizing performance through consolidation strategies like data caching and connection pooling occurs internally to the data server, the use of XML as the format does not affect this function.

Data servers can leverage XML messaging protocols and XML data formats

An XML-enabled data server supports XML as the unified data access format. When an application submits a request to the data server, the data server fulfills it with an XML document. Given the rise of XML messaging discussed in Chapter 4, the data server should probably support this interaction over SOAP, using an interface specified in WSDL. Merely retrieving ad hoc bits of data as XML documents that the application then has to translate into programming data structures doesn't add much benefit. Programmatic solutions such as **ODBC** and **JDBC** already satisfy this need. The more substantial benefit comes from defining synthetic XML documents that form customized packages of data suited to a particular purpose.

They should support mapping from DBMS to XML Schema

To deliver a synthetic XML document, the data server must have a mapping between the document type and the structures managed by backend DBMSs. A developer defines an XML DTD or Schema for the document type and then maps fields in the various database schemas to element and attribute types. The developer also defines the keys used to select the correct records for populating a document instance. At runtime, an application submits a request for a synthetic document type and the appropriate keys. The data server then looks up the mapping, constructs queries based on the mapping and the keys, and puts the results into an XML document. This results document is valid with respect to the specified DTD or Schema.

Some DBMSs include basic data server features

In some cases, a DBMS vendor may include some data server capabilities with its DBMS product. For instance, Oracle9i includes XML mapping capabilities. In cases where the need for a data sever stems from a small set of homogenous databases attempting to serve many different applications, this solution is sufficient. But when the need for a data server stems from a set of applications attempting to aggregate data across heterogeneous databases, you probably need a separate data server product.

Native XML store and data server markets are merging

Such products include eXcelon's eXtensible Information Server and Versant enJin, both of which are based on object persistence engines. Data servers require many of the capabilities of backend databases to provide high availability and transactional integrity. They use their own persistence engine as a staging area between applications and backend DBMSs. Therefore, most of the native XML store products discussed previously can also operate as XML data servers by adding features for synchronizing with backend databases. In fact, many vendors of these products are finding that this approach drives a substantial percentage of their sales. Conversely, data server products like eXcelon and enJin can operate as native XML stores, so distinctions between the two markets are blurring. When evaluating either type of product's suitability as a data server, focus on the facilities for mapping backend data to XML documents and the efficiency of performance optimization strategies like caching and pooling.

Application Servers

Application servers are hubs of XML document flow

Application servers operate in the middle tier, applying business logic to data, then handing off the results for presentation. In this capacity, they have three primary reasons for working with XML documents.

1. They may need to accept data as XML documents from data servers.
2. They may need to provide business results as XML documents to content servers.
3. They may have to exchange XML-formatted business messages with other application servers.

To support these operations, the application server can supply basic and advanced services.

Basic services include XML, XSLT, and SOAP support

Basic services include the execution of XML and XSLT processors, as well as a SOAP implementation. Whether it extracts data from XML documents, exchanges XML business documents, or produces XML business results, the application server needs the access and creation capabilities of an XML processor. Because many developers use XSLT for pre- and postdocument processing, support for this standard should be part of the basic package. Interaction with XML-enabled data, application, and content servers almost certainly includes SOAP communication, so an implementation of the protocol is essential.

Basic services must be tested, optimized, and supported

Theoretically, because an application server can execute any code in a language it supports, providing basic services is simply a matter of downloading XML and XLST processors plus a SOAP implementation, then installing them. Practically, assuring the performance and quality of execution requires the vendor to at the very least certify components for use with the application server and probably include the recommended packages in the product distribution. You want to make sure that the vendor has tested the particular components, can provide estimates of how much throughput these components can handle, and knows how to support their use with its application server. For **J2EE** application servers, most vendors recommend the Xerces XML processor, the Xalan XSLT processor, and

have chosen either their own or a particular third-party SOAP implementation. Microsoft has its own XML processor, XSLT processor, and SOAP implementation for its application server products.

Abstraction frameworks improve flexibility

Advanced services tend to vary significantly across application servers and evolve rapidly over time. Therefore, it's more appropriate to focus on the categories of advanced services rather than particular instances. Most advanced services are delivered in the form of frameworks. There are abstraction frameworks and task frameworks. Abstraction frameworks give developers more flexibility to make future changes by performing operations at a higher level. Two excellent examples are Sun's Java API for XML Processing (JAXP) and Java API for XML Messaging (JAXM). Both of these frameworks provide high-level APIs for performing specific XML-related operations. By programming to these abstract APIs rather than the concrete APIs of specific components, developers make it possible to easily switch their XML processor or XML messaging protocol.

Task frameworks improve productivity

Task frameworks provide additional functionality for building specific types of applications. Personalization is a good example of a task framework used to produce XML documents for content servers. These types of applications use metadata about user preferences and metadata about content topics to generate customized content. Because XML is a convenient format for both types of metadata, there is the opportunity to deliver a package that greatly simplifies the development of such applications. But perhaps the best XML-related example of such an application is B2B messaging. This type of application touches on a host of issues, from specifying the allowable flows of messages, to generating views of executing processes, to integrating with back-end systems. Providing all this functionality would be difficult for a single application development

team. By using XML, vendors can deliver a widely applicable framework that puts such applications within the reach of more organizations. All the major application server vendors—including BEA, IBM, Microsoft, Oracle, and Sun—provide their own flavors of both personalization and B2B messaging frameworks.

Content Servers

Content servers enhance the Web user experience

Content servers combine data from DBMSs, results from business operations, and authored content into presentation formats suitable for different users. XML-based technologies improve every stage of the fulfillment pipeline. At the very end of the pipeline, they enable dynamic layouts that better fit each user's needs. In the middle of the pipeline, they make it easier to connect a user to the exact information he wants. At the beginning of the pipeline, they make it easier to acquire the library of content necessary to satisfy the user base. Most content servers focus on one or two aspects of this pipeline, so implementing a complete XML content strategy may require several types of content servers.

They can dynamically generate presentation layout

The most common use for XML in content servers is applying dynamic presentation to XML content. This process occurs as described in Chapter 3's discussion of using XSLT to generate pages in XML-based presentation languages such as HTML, VoiceXML, and WML. Based on variables, including the type of client device, the type of content, and the localization settings for the user, the content server selects an XSLT transform and applies it to the XML document. Because most Web servers have programming extensions that support XSLT, you won't need any additional server infrastructure if all you want is dynamic presentation.

XML provides metadata cues for searching

Customizing layouts for users is only part of the content delivery equation. Users also need help finding the content that addresses their immediate needs. Traditional search engines suffer from the

problem (raised in Chapter 1) of distinguishing between different contexts for the same word. With XML content, a search engine can use the element structure and attribute values to improve search precision. Using an XML-aware search engine helps maximize the benefits of an XML-based content strategy. Usually, employing such a product involves assigning a dedicated server or cluster of servers to perform searches that then refer users to the appropriate content. Such standalone solutions include DocSoft's extend XML and XML Global's GoXML Search. Of course, most of the CMS and native XML store products discussed previously can perform searches on XML document collections, but this approach works only if you store all the content you plan to search in one of these products.

The Semantic Web uses metadata at the site level

XML-aware search engines leverage metadata at the element and attribute levels. However, metadata can also apply to entire collections of content. The foundation of the *Semantic Web* is the use of metadata to provide a conceptual map of an entire site or group of sites. Another W3C Recommendation, **Resource Definition Framework (RDF)**, provides a standardized XML vocabulary for describing the types of content offered, the relationships among content, and the conditions under which content might be relevant. Most site creators use an implicit information model in selecting and organizing content. RDF makes it possible to explicitly state this model. The availability of machine-readable models facilitates automated information retrieval, filtering, and visualization capabilities far beyond those of traditional search engines. The Semantic Web is in its early development and much of the work is in the form of research and open source projects. However, in the near future, RDF may migrate into mainstream content infrastructure. Web servers will offer RDF descriptions. Search engines will use these descriptions as part of the search criteria. Authoring tools will generate these descriptions.

Structured content facilitates distributed collaboration

In addition to making it easier to find content, XML also makes it easier to acquire content. Content can come from two sources: You can create it, or you can borrow it. When creating content, the ability of multiple authors to effectively collaborate greatly enhances productivity. **Web Distributed Authoring and Versioning** (WebDAV), a set of XML-based extensions to HTTP from the **IETF**, makes it possible for authors to work together to create, enhance, and maintain content. A WebDAV server manages contributions, tracks changes, and enforces permissions. A number of portal servers, including Microsoft's SharePoint Portal Server and Oracle9iAS Portal, use WebDAV to enable the collaborative editing of portal content. Common Web servers such as Apache and IIS also support WebDAV. Any client that speaks the WebDAV protocol can use these servers to collaborate on documents. Such clients include popular content authoring tools such as Adobe Acrobat and Microsoft Office. Taken to an extreme, WebDAV enables the replacement of traditional document management systems with a set of distributed WebDAV-capable servers. Oracle iFS and Xythos's Web File Server use this approach.

Syndication benefits from both structured content and standard protocols

It is often more cost effective to borrow content from someone else than generate it yourself. However, this type of syndication faces two problems. First, it is often difficult to fit third party content into an application because of differences in layout. XML solves this problem by giving both parties a format for exchanging information separate from presentation. The subscriber knows the structure of each publisher's content, so it can use XSLT to integrate content from different sources and apply its preferred layout. There is also the problem of how to automatically negotiate subscriptions, track usage, and update information. **Information and Content Exchange (ICE)** addresses these issues by providing a standard XML protocol for such interactions between subscribers and publishers. ICE support is available in a wide variety of products that

generate and manage content, including Interwoven's OpenSyndicate, Oracle9i, and Vignette's Content Syndication Server.

Data-Oriented Components

Developers need XML-capable components

Deploying XML-based systems requires that developers write software that accesses, manipulates, and creates XML as if it were data. In some cases, this software runs on top of the server infrastructure discussed previously. In other cases, the software may operate as a standalone program. In all cases, XML offers improved productivity through the wide availability of components for generating, processing, and distributing machine-readable XML data, which developers require.

Points of leverage include development, transformation, and distribution

While there are numerous types of utilities that can assist them, there are three categories crucial to most serious applications. Any serious XML programming requires moving XML data in and out of internal programming data structures. Development tools reduce the amount of coding and improve the quality of code for this repetitious task. In practice, using XML as data also requires transforming documents from one XML format to another, as well as between XML and other formats. Transformation tools simplify the problem of creating XSLT transforms and integrating external data into XML architectures. Moreover, most data-oriented applications of XML involve distributed communication. While application servers and packaged solutions provide basic functionality, the Web services paradigm often requires either integrating a variety of server-provided services or deploying standalone functionality to nonserver devices. In these cases Web services components reduce time to market.

Development Tools

Most IDEs offer basic document interaction features

The most common task in developing XML application is ensuring that a piece of code that manipulates an XML document works as desired. Performing this task requires the developer to work

simultaneously with a document and the associated code. It helps the developer to view the XML document in two different ways. First, developers need a text view but with all the additional features they've come to expect from a code editor—indenting, highlighting, and syntax checking. Second, they need a browsable tree view, much like they have for class hierarchies. Together these two views allow the developer to maintain a good mental model of the document and thereby understand how the code running against it should work. Many commercial **IDEs**, especially those for Java and from Microsoft, provide this functionality. There are also open source modules for adding the text viewing capabilities to popular code editors like **EMACS**.

Schema compilers generate XML manipulation code

While IDEs supply features that help when working on code that manipulates XML documents, there is the issue of writing this code in the first place. Many XML applications work with a known set of XML formats. For example, an application for the insurance industry would work with documents conforming to insurance-related XML DTDs or Schemas. Writing the code to move data back and forth between documents using these formats and internal data structure is time consuming and repetitive. Schema compilers can take a DTD or Schema and generate much of this code, saving time and improving quality. Such tools include Breeze Factor's Breeze XML Studio (Java), Bristol Technology's eXactML (C++), Microsoft's Visual Studio .NET (C++ and Visual Basic), The Mind Electric's Electric XML+, OracleXML Class Generator (C++ and Java), and Sun's Java Architecture for XML Binding (JAXB).

Transformation Tools

Mapping tools help generate XSLT data transforms

Many applications concern the exchange of information about coherent business entities. It is likely that different parties to these exchanges will have different definitions of these entities. These

different definitions will probably take the form of different DTDs or Schemas. As you saw in Chapter 3, XSLT can address this problem. However, for large applications that work with many different formats, you may find it difficult and time consuming to create all the necessary XSLT by hand. Creating a supply chain management system may require writing transforms for hundreds of formats. Moreover, these formats can change regularly. Developers need tools to make this process more efficient. A number of vendors offer visual mapping tools that let the developer indicate how to move element and attribute values from the source format to element and attribute values of the target format. The tools then generate the XSLT code, and many provide debugging features for making sure the generated code works properly. Products in this category include ActiveState's Visual XSLT (Visual .NET plug-in), B-Bop Xfinity Designer, eXcelon's Stylus Studio, IBM's Visual XML Transformation Tool, Induslogic's XSL Stylesheet Editor, and TIBCO XMLTransform.

Integration tools move data between XML and other data formats

Unfortunately, not all data exists in an XML format. Many applications must map data from non-XML sources to XML formats and vice versa. Developers need components that can perform this mapping both in bulk and upon request. These tools are similar to those for mapping data from relational databases to object-oriented programming structures and migrating data from operational stores to data warehouses. They include features for browsing the structures of data sources and then specifying how to translate from data fields to elements and attributes. These tools are quite sophisticated because processing data from a non-XML source may require more than simply pulling data from a column and putting in an element. There are also datatype conversions and collapsing or exploding relationships between fields. The two leading products for this purpose are Data Junction's Integration Engine and Mercator's Integration Broker.

Distributed Application Components

Large vendors offer complete distributed XML paradigms

As previewed in the discussion of Application Servers in the preceding Server Infrastructure section, many applications use XML to facilitate distributed communication in custom applications. Usually, such applications use SOAP and the other Web services protocols. Task frameworks can help improve productivity, but the issues can extend beyond the boundaries of application servers. It often helps to have tight integration of distributed application development across development tools, frameworks, and servers. The development tools can then quickly lead developers in the use of the advanced framework and server features. This breadth of integration usually comes under the umbrella of a large vendor's overarching distributed XML paradigm. Examples include BEA WebLogic Workshop, IBM WebServices, Microsoft .NET, and Sun ONE.

You can choose between integrated and best-of-breed approaches

In many cases, you will already have a substantial commitment to a vendor through existing server infrastructure and development tools. Obviously, the least disruptive choice for Web services components will be that vendor. However, if you plan to make a decision on your preferred supplier based on Web services support, you can go either the integrated solution or best-of-breed route. The integrated solution route offers the benefit of making a single vendor responsible for the success of your applications. It has the drawback of restricting you to that vendor. The best-of-breed route has the advantage of enabling you to choose the best vendor for other infrastructure such as platforms and tools. It has the disadvantage of making you responsible for integrating the entire system. Major vendors offer integration along two dimensions: platforms and tools. In this case, the platform is a composite of hardware architecture, operating system, and mission critical components like DBMSs.

Integration occurs in both platforms and tools

Figure 5-6 presents a conceptual graph of where major vendor offerings fall in the space formed by these two dimensions. Microsoft has the highest platform specificity because it requires that you use its operating system and other server infrastructure. IBM and Sun both offer their solutions on multiple platforms but, of course, they try to drive the sales of their specific platforms. BEA is almost completely platform agnostic. All vendors try to take advantage of a Web service's potential productivity gains to some extent with a certain degree of development tool integration. BEA does this through a high-level framework but leaves core programming tool choices. IBM and Sun offer their own IDEs based on standard programming and scripting languages. Microsoft offers its own IDEs and languages.

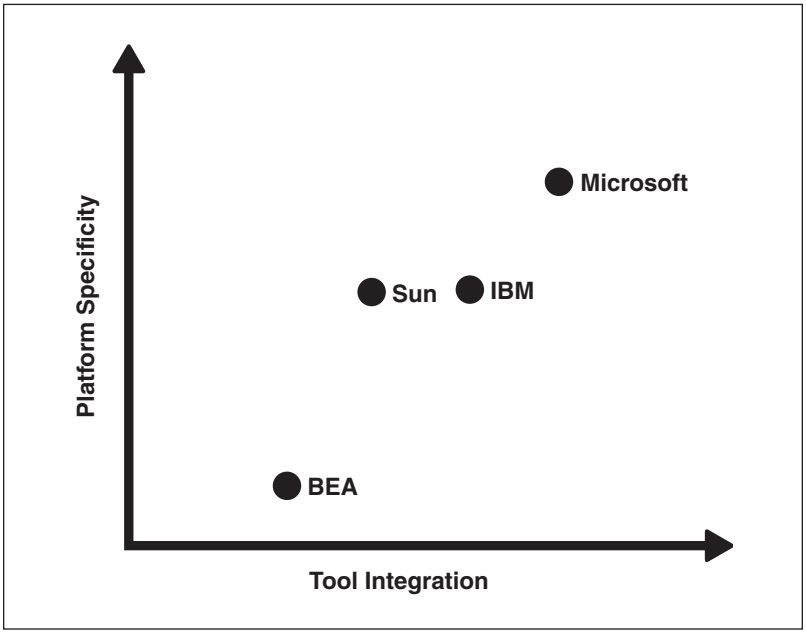

Figure 5-6: Degree of Integration by Major XML Component Vendors

Small vendors offer modular small footprint solutions

The large vendor initiatives have drawbacks. To achieve the desired breadth of integration, they often sacrifice modularity and performance. Obviously, they want developers to buy the complete packages, so they often require all-or-nothing commitments. Maintaining the synchronization between all the components usually introduces a substantial amount of overhead. These drawbacks are severe for applications not deployed in application servers, especially if targeted at small devices. Smaller products such as CapeClear's CapeStudio, The Mind Electric's GLUE, and Polar Lake fill this gap with components that Web service enable application-specific code in a tight, standalone footprint.

Content-Oriented Components

Authors need XML-capable components

Delivering XML content to users requires off-the-shelf components for authoring, presenting, and delivering human-readable XML content. The principal benefit of XML as a content standard is that content from all sources uses the same infrastructure. Different groups of authors may contribute content, and it is easy to convert data from applications into content for users. Unfortunately, this capability raises the complexity of the authoring task because authors need to seamlessly merge static content from numerous contributors with dynamically generated content from applications. Well-defined processes become critical to coordinating multiple content production paths.

Points of leverage include authoring, layout, and management

There are three things authors must do to take advantage of XML's benefits for content delivery: (1) format content as XML documents, (2) create layouts for this content, and (3) follow a rigorous production process to ensure uniform quality. Authoring tools, layout tools, and content management components provide the functionality necessary to quickly deliver high-quality, high-appeal content.

Authoring Tools

Authoring tools speed the input of content as XML

Much XML content includes static documents produced by human authors. Creating them with a text editor is a slow and error-prone process. Moreover, many documents have primarily static content with select pieces of information extracted dynamically from an application. Authors certainly do not want to have to manually enter programming-related tags to drive this capture process. Document authoring tools offer three primary features for improving productivity.

They can offer ad hoc as well as structured input modes

First, they offer word-processing-like interfaces that enable authors to create ad hoc documents. This feature separates the free text information from the layout. Second, they offer a wizard- or form-based interface that authors can use to populate documents conforming to a more detailed DTD or Schema. This feature speeds data entry for content such as customer contact reports. Finally, they include features for specify placeholders for application content that a content server uses to insert dynamic values at delivery time. This feature requires integration with a runtime content processing engine. While different products offer different mixes of these three features and are evolving quickly, some of the most popular products include Arbortext's Epic Editor, XMLSpy's Document Editor, and SoftQuad's XMetaL.

Layout Tools

Layout tools merge graphical layout with XSL generation

XSL is a highly sophisticated layout description language. A large proportion of layout designers with the necessary graphical design background may not have the technical background necessary to create stylesheets by hand. Therefore, they need stylesheet layout tools for automatically configuring different page regions to display different types of elements and specifying the text formatting based on rules such as element type and attribute values. These tools must also be aware of DTDs and Schema so that authors can match detailed layouts with detailed formats.

Typically, layout tools allow designers to work in either concrete or abstract modes. In a concrete mode, the designer has an example XML content document and applies formatting to that document's information. The tool then abstracts this information to generate a generic stylesheet. By using several example content documents, the author can ensure complete coverage of different possible cases. In an abstract mode, the designer applies formatting rules to a DTD or Schema. This approach gives the author more flexibility in dynamically determining how to present information based on its value, but it is not as intuitive. Before selecting a layout tool, you should work with your designers to decide which mode is more important. The best tools allow designers to switch between modes, but they still tend to emphasize one over the other. Leading products include Arbortext's Epic Editor, eXcelon's Stylus Studio, IBM's XSL Stylesheet Editor, Whitehill <xsl>Composer and XMLSpy's XSLT Designer.

Content Management Components

Managing the content production process is an issue

As an organization adopts XML, more and more people become involved in authoring XML content. As with HTML and SGML, managing this content poses a logistical challenge. Moreover, because software will automatically generate many documents, there is the possibility for new challenges and even greater content volume. Figure 5-7 shows the generic architecture of a CMS that can help address this problem. As Figure 5-7 shows, CMSs typically have several major components.

- *Repository.* The repository provides a robust and fault-tolerant location for storing XML documents, stylesheets, and DTDs or Schema. It consists of an interface that enables the content management system to store and retrieve information, a manager that controls storage mechanisms, and the storage

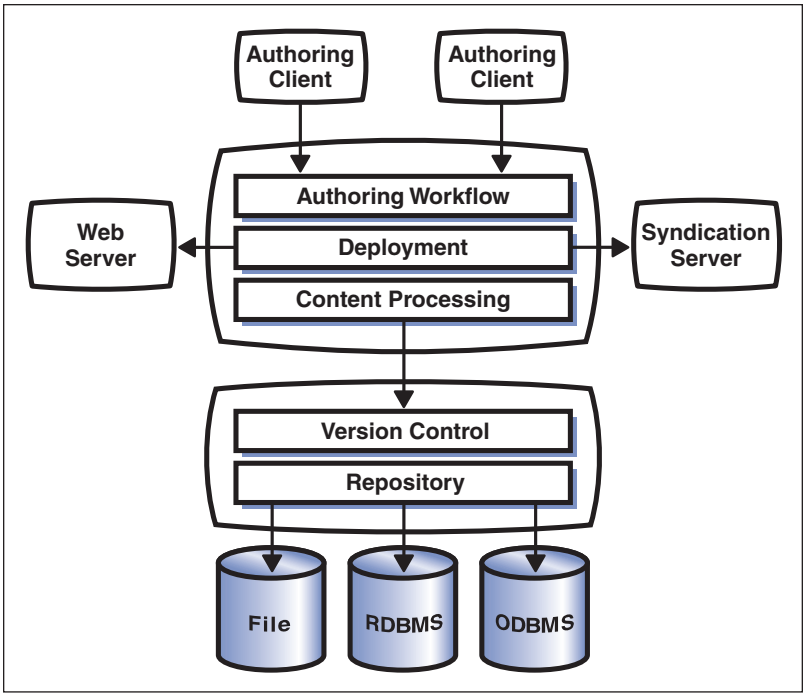

Figure 5-7: Generic CMS Architecture

mechanisms themselves. These mechanisms may include filesystems, relational databases, or object databases.

- *Version control.* The version control subsystem performs two functions. First, it prevents multiple authors from simultaneously making changes in the same content. Second, it maintains a version tree of all content. All requests to store or retrieve content must go through the version control system because it maintains the mapping of logical versions to physical data. CMSs that support WebDAV for version control offer the advantage of easier interoperability with different types of authoring clients.
- *Authoring workflow.* A content management system needs a component to coordinate the contributions and revision

process. Typically, this coordination includes the maintenance of an authoring schedule and assignments for each author. It ensures the routing of documents from one author to another based on content dependencies. As published content reflects on the organization, this routing may also include approval and revision workflow.

- *Content processing.* Once authors have submitted content, the CMS may offer a number of processing functions. Foremost among these functions is content indexing. If you commit to managing all your content within a CMS, you rely on the search functions it provides. To perform such searches efficiently, authors need to specify how to index the content. In cases where documents include dynamic information bound to application data, another processing function includes accessing, formating, and distributing this data. Some CMSs offer advanced filtering and transformation processing to automatically create different views of content suitable for different audiences. For example, a filter might specify how to automatically generate a summary view for a particular type of document.
- *Deployment management.* When content becomes ready for consumption, an administrator must release it for distribution. Depending on the channel, managing this deployment may take several forms. For static content intended for Web servers, it might create the directory structure and install it in all the Web server machines. For dynamic content intended for Web servers, it may require a substantial amount of configuration information governing network topology, access control, and performance parameters. In cases where you intend to syndicate your content, the CMS will need additional information to manage this process.

These are many factors to consider in choosing a CMS

There are a wide variety of CMSs, each with its own target use. Content management is a complicated topic, and the choice of

product requires considerable analysis. Factors to consider include the primacy of Web over other channels, the use of non-XML formats, and integration with dynamic data sources. Products to consider include Arbortext's Epic E-Content Engine (no repository), BroadVision Publishing Center, Chrystal's Astoria, Documentum4i, Interwoven TeamSite, OmniMark Technologies'/OmniMark, Red Bridge Interactive's DynaBase, SiberLogic's SiberSafe, and Vignette's Content Suite.

Design Tools

The design of an XML format determines the intent of documents

The previous two sections divided components into data and content manipulation categories. How do you know if XML is data or content? It depends on the design of the XML formats used by your system. When you plan to use formats that interact primarily with application code or database servers, you design them to reflect the rigid structures and datatype focus used by these systems. When you plan to use formats that primarily drive the organization of information delivered to users, you design them with flexible structures and semantic focus used by people. These decisions actually occur early in the development process and determine how the entire team uses XML.

Design tools let you focus on requirements instead of syntax

In the case of data-oriented formats, the design process is similar to that for programming data structures or database schemas. In the case of content-oriented formats, the design process is similar to information design or user requirements analysis. In either case, the designer needs to focus on the structure of information rather than the syntax for DTDs or Schema. Design tools facilitate the conceptual design of documents and then generate the appropriate syntax. A graphical designer works much like a data modeling tool. Designers use a tree metaphor to create the basic element structure and then use dialog boxes to configure the allowable element and

attributes data. In practice, the generated syntax may need manual tuning, especially for very complex formats, but tools can usually cover most of the cases.

Choosing a product comes down to personal preference

There are currently two primary choices for graphical design tools: Altova's XML Spy 4.2 Integrated Development Environment (IDE) and TIBCO TurboXML. On paper, they provide much the same feature set. Both provide tree, inspector, and syntax views of XML formats. Both support both DTDs and Schema. In fact, many organizations use these tools to convert their existing DTDs to Schema. Both do a pretty good job of generating the correct syntax. In practice, they provide interfaces that are different enough to yield distinct experiences. Many designers develop a strong preference for one interface over another after using them, so you should thoroughly evaluate both before choosing one.

Infrastructure Selection Strategy

You may have to decide on infrastructure early in the process

This chapter presented XML software infrastructure in the order that you would bring pieces together during development. To maintain maximum flexibility, you would actually make your product selections in the same order. However, the practicalities of the procurement process will probably force you to make decisions early in the project, or perhaps even before starting a specific project. Also, as you will see in Chapter 6, establishing your infrastructure beforehand can also make it easier to acquire staff with the appropriate skills.

Application server choice will drive data-oriented infrastructure

If you do have to make an early decision on XML software infrastructure, it's important to know whether you plan to use XML for data-oriented or content-oriented projects. For data-oriented projects, the choice of application server will usually drive the rest of the decisions. Your existing environment, price, and the availability

of applicable features like Web services frameworks should factor most prominently. This choice will probably make the selection of fundamental components for you. It will also influence whether you need other server infrastructure and data-oriented components, as well as the most suitable products if you do.

CMS choice will drive content-oriented infrastructure

For content-oriented projects, the choice of CMS will usually drive the rest of the decisions. In nearly all cases, the CMS will include fundamental and other content-oriented components. You may have to choose a data server separately, depending on the degree and volume of dynamic content generation. In cases where you have a balance of data-oriented and content-oriented development, you will have to simultaneously decide on an application server and CMS, taking into account the success of other organizations successfully using different combinations.