Distribution Alternatives for Superimposed Information Services in Digital Libraries

Sudarshan Murthy‡, David Maier∗, Lois Delcambre∗

Department of Computer Science, Portland State University∆
PO Box 751 Portland, OR 97207-0751 USA
{smurthy, maier, lmd}@cs.pdx.edu
http://datalab.cs.pdx.edu/sparce

Abstract. Component-based, service-oriented digital library architectures are being used to provide superimposed information services such as annotations. Although much attention has been paid to the issues in building components for these services, not enough attention has been paid to their deployment—specifically to distribution. We believe that matching the location of executable and data components to the needs of patrons and digital libraries can improve the overall system performance. We describe five distribution alternatives for providing superimposed information services in a digital library and discuss the trade-offs for each alternative. We also define some metrics to compare the performance of the alternatives. We use our middleware architecture for superimposed information management, called the Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE), for illustration.

1 Introduction

Our research on superimposed information focuses on allowing users to superimpose [6] new information such as annotations and summaries on top of existing base information such as web pages and PDF documents. In addition to superimposing annotations, a user may select parts of existing information and create new linkages among those selections. For example, a user may create an alternative organization of sections in a PDF document, or the user may create a table of selected contents.

Figure 1 (a) shows superimposed information elements that were created and organized in one of our superimposed applications called RIDPad [8]. It shows four items labeled ‘Statement’, ‘FONSI’, ‘Details’ and ‘Issues,’ each linked to a selection in other documents such as spreadsheets and word processor documents. For example, the item labeled ‘Issues’ is linked to a selection in an MS Excel spreadsheet; the item labeled ‘FONSI’ is linked to a selection in a MS Word document (shown in Figure 1 (b)). The box labeled ‘Decision’ groups items. RIDPad works with a com-

‡Author’s work has been supported by US NSF grant IIS 0086002.
∗Author’s work has been supported by US NSF grant IIS 9817492.
∆This work was originally done when the authors were affiliated with the Department of Computer Science and Engineering, OGI School of Science and Engineering at OHSU.
ponent-based, middleware architecture called the Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE) [8].

Some digital Library (DL) systems also use component-based architectures [13, 15] to support creation of annotations and metadata over information in a DL. Some of the benefits of component-based architectures are that architectural components may be replaced with alternatives and new components may be plugged in easily. They make it easier to build new services using component stacks.

Another benefit of component-based architectures, one that does not receive as much attention, is flexibility of deployment. With proper interface design and abstraction, components (both data and executable) may be either centrally-deployed or distributed, without affecting the services they provide. This flexibility is important because placing a component at the right location can improve performance, especially for frequently used services.

![Fig. 1. (a) Superimposed information organized using RIDPad, a superimposed application. (b) The MS Word selection linked to the item FONSI of the RIDPad document activated.](image)

In this paper, we present five distribution alternatives and their trade-offs when providing superimposed information services, such as annotations, in a DL. We use our component-based middleware architecture SPARCE to illustrate the alternatives. We present the alternatives without a specific DL architecture in mind because they should apply to component-based DL architectures in general.

To motivate, we present a hypothetical annotation system in Figure 2 (fashioned after Open Digital Library [11]) and outline two distribution alternatives for it. This system has three components: user interface, annotation, and an archive. One distribution alternative, call it Alternative A, is to run the user interface and the annotation components on a patron’s (client’s) computer, and run the archive component within a DL server. Alternative B is to run the user interface on a patron’s computer, and run the other two components within a DL server. These alternatives could differ significantly in maintainability and performance. For example, Alternative B may be more maintainable because few components run outside the DL server, but it has the potential to increase the load on the server. The alternatives differ also in the interface a DL server needs to provide to the outside world. Alternative A requires a DL server to
provide interface to the Archive component (the Annotate component connects from the patron’s computer to the Archive component in the DL server), whereas Alternative B requires the server to provide interface to the Annotate component.

![Alternative A](image)

**Fig. 2.** An ODL-style annotation system. Each dotted rectangle contains components collocated in the distribution alternative called out

Proximity of components can affect the overall system performance. To give an idea, we present results from a simple experiment we conducted. Table 1 shows the mean round-trip time for three web-service methods based on SOAP [14] bound to HTTP. From the table we see that the mean round-trip time over a WAN is about 60 times that over a LAN when sending and receiving 400 bytes. These numbers tell us that performance could be improved by placing components with a higher number of round trips between them closer to each other (based on network distance, not geographic distance). For example, if we know that the Annotate component makes many round trips to the Archive component to serve a single request from the user interface, we may benefit by collocating the Annotate and Archive components in a DL server as in Alternative B.

**Table 1.** Mean round-trip time for SOAP-based web service methods via HTTP. Columns Input and Output denote the number and type of inputs and outputs respectively for the methods; Column Local shows round-trip time when client and server are on the same computer, LAN shows round-trip time when client and server are connected over a LAN (100 MBPS, one hop), WAN shows round-trip time when the client and server are connected over a WAN (756 KBPS DSL connection, more than 18 hops)

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Mean round-trip time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Local</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
<td>2.94</td>
</tr>
<tr>
<td>10 integers</td>
<td>10 integers</td>
<td>3.13</td>
</tr>
<tr>
<td>One 400-byte array</td>
<td>One 400-byte array</td>
<td>7.87</td>
</tr>
</tbody>
</table>

The numbers in the last column of Table 1 deserve some clarification, because they indicate rather large round-trip times. A WAN communication is heavily influenced by network conditions (for example, congestion), policies, and operations such
as routing and filtering (for example, firewalls). Table 1 shows numbers for communication between a computer in a home office (client) and a computer inside OGI (server). In this setting, all communication passes through the policies and infrastructure of at least four parties: the client, the client’s ISP, OGI, and us (the web service provider). Additionally, all packets are subject to inspection at each hop. Finally, one web service roundtrip might require more than one roundtrip at lower network layers.

The rest of this paper is organized as follows. Sections 2 and 3 give an overview of superimposed information and SPARCE, respectively. Section 4 defines some metrics, and details five distribution alternatives and their trade-offs. Section 5 discusses some issues in employing those distribution alternatives. Section 6 provides a brief overview of related work. Section 7 concludes the paper.

2 Superimposed Information

Superimposed information refers to data placed over existing information sources to help select, access, organize, connect, and reuse information elements in those sources [6]. Existing information sources reside in the base layer, and data placed over one or more base sources resides in the superimposed layer (see Figure 3). Word-processor documents, databases, and web pages are examples of base documents. A stand-off annotation is an example of superimposed information (because it is stored separately from the object of annotation, and it maintains a link to the object). An application that manipulates base information is called a base application; an application that manipulates superimposed information is called a superimposed application. Adobe Acrobat is a base application for PDF documents, and RIDPad (Figure 1 (a)) is a superimposed application.

![Fig. 3. Layers of information in a superimposed information management system. A mark connects a superimposed information element to the base layer](image)

2.1 Marks

A superimposed information element (such as an annotation) can reference a base information element (such as a selection in a spreadsheet) using an abstraction called
a mark. Figure 3 shows the superimposed layer using marks to address base elements. We have implemented the mark abstraction for base types such as PDF, HTML, and MS Word. The addressing scheme a mark implementation uses often depends on the base type(s) it supports. For example, an implementation for PDF documents may use page number and index of the first and last words in a text selection, whereas an implementation for XML documents may use XPath. All mark implementations provide a common interface to address base information, regardless of the base types or access protocols they support. A superimposed application can work uniformly with any base type by virtue of this common interface. For example, the items in RIDPad document of Figure 1 (a) refer to marks in MS Word and MS Excel documents, but the RIDPad application works uniformly with all marks. Figure 1 (b) shows the MS Word mark of the ‘FONSI’ item activated.

2.2 Excerpts and Contexts

Superimposed applications may sometimes need to incorporate the content of base-layer elements in the superimposed layer. For example, an application might use the extracted base-layer content as the label of a superimposed element. We call the contents of a base-layer element an excerpt. An excerpt can be of various types. For example, it may be text or an image. An excerpt of one type might also be transformed into other types. For example, formatted text in a word processor could also be seen as plain text, or as a graphical image.

Fig. 4. The context of the MS Word mark of the item FONSI of the example RIDPad document in the Context Browser. The browser is showing part of the HTML markup required to display the excerpt for the mark formatted as is in the base layer (the highlighted region of Figure 1 (b)).

In addition to excerpts, superimposed applications may use other information related to base-layer elements. For example, an application may group superimposed information by the section in which the base-layer elements reside. To do so, the application needs to retrieve the section heading (assuming one exists) of each base-layer element. We call information concerning a base-layer element, retrieved from the base layer, its context. Presentation information such as font name and location information such as line number might be included in the context of a mark. Each
such piece of information is a context element, and context is a collection of context elements. Because we use the same mechanism to support both contexts and excerpts, we often use the term “context” broadly to refer to both kinds of information about a base-layer element. Figure 4 shows the context of a MS Word mark in a browser (for the mark activated in Figure 1 (b)).

3 SPARCE

The Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE) is a middleware-based approach for mark and context management [8]. It is designed to be extensible in terms of supporting new base-layer types and context-element types, without adversely affecting existing superimposed applications. Figure 5 shows the SPARCE reference model. The Mark Manager provides operations such as creating marks and storing them in a marks repository that it maintains. The Context Manager retrieves context information for a mark. Superimposed applications use the managers to create marks and access context (which in turn use base applications).

The Mark Manager supports three operations for marks: creation, retrieval, and activation. Mark creation is the operation of generating a new mark corresponding to a selection in a base layer. This operation consists of three steps: generating the address of base information (and other auxiliary information), using the information generated to create a mark object, and storing the mark object in the mark repository. Details of each mark, such as the address of the base document and the selection inside it, are stored as an XML file. The mark repository is a database of such XML files.

The mark retrieval operation returns a mark from the mark repository. Mark activation is the operation of navigating to a location inside the base layer, using the information supplied by a mark. SPARCE uses mediators called context agents to retrieve context information for a mark from the base layer. A context agent interacts with a base application to retrieve context. The name of the context agent to use for each mark instance is one of the details stored in the mark repository. SPARCE uses this information to instantiate an appropriate context agent for a mark instance. A superimposed application receives a reference to the instance of context agent from SPARCE, and then works directly with the agent instance to retrieve context.

The components of SPARCE (see Figure 5) may be mapped to those of the ODL-style annotation system we introduced in Figure 2. Superimposed applications (patron applications) provide the user interface. These could be desktop applications or browser-based applications (for example, Java applets) running on a patron’s computer. The Mark Manager, the Context Manager, and the base applications constitute the Annotate component. The base documents and a DL’s interface to access them (if required) roughly constitute the Archive component.
Fig. 5. The SPARCE reference model. Solid arrows show dependency, dotted arrows show data flow (not all data-flows shown). The dashed lines partition components to correspond to the blocks in Figure 2

4 Distribution Alternatives

All components of SPARCE shown in Figure 5 are candidates for distribution, but we consider only four components: the patron (superimposed) application, the Mark Manager, the Context Manager, and base applications. We vary the location of these components and describe five distribution alternatives for SPARCE (see Figures 7, 8, and 9) when providing superimposed information services in a DL. For simplicity, we assume the following configuration for all alternatives:

- The DL server contains base documents (and that an appropriate interface in the DL server is used to access the documents).
- The two manager modules (the Mark Manager and the Context Manager) run on the same computer.
- The mark repository is stored wherever the Mark Manager is deployed.
- The patron application always runs on a patron's computer.
- The superimposed information is stored on a patron's computer.

We first present a goal for distribution and some related metrics. We do not provide results based on these metrics, but estimate trends based on a few rules of thumb (see Section 4.7). We assume a patron uses a high-speed Internet connection (such as broadband) to a DL server. We also assume that the ratios of the mean round-trip times shown in the third row (for 400-byte array input and output) of Table 1 hold.

A note on terminology: the term “patron’s computer” in the rest of this paper may mean a stand-alone computer or a computer in a network local to the patron. Our description of alternatives would be valid for either interpretation of the term.
4.1 Goals and Metrics

Several metrics such as latency (for example, time to serve a request), load (for example, the number of active processes) and throughput (for example, the number of requests processed per unit time) should be considered for a thorough analysis of the alternatives. However, we discuss only latency in detail. Section 5 touches on issues such as load.

The latency of a patron application’s request \( T_{pm} \) is the duration between the patron initiating a request in the patron application (to a manager module) and the patron receiving a corresponding response. It is made up of the following components (see Figure 6):

- \( t_{bb} \): The time taken by a base application to complete a requested operation. An example is the time to retrieve a context element’s value.
- \( t_{mb} \): The round-trip time between a manager module and a base application. This time measures the duration between a manager module receiving a request from a patron application and the manager module returning a corresponding response, after discounting the time the base application takes to complete its work.
- \( t_{pm} \): The round-trip time between a patron application and a manager module. This time measures the duration between the patron initiating a request in the patron application (to a manager module) and the patron receiving a corresponding response, after discounting the time the manager module and the base application need to complete their work.

![Fig. 6. Components of latency of a patron application’s request. The subscripts p, m, and b stand for patron application, manager module, and base application respectively. They identify the pair of architectural components with which a latency term is associated.](image)

In distributing the architectural components, one of our goals is to minimize the latency of a patron application’s request \( T_{pm} \). Because this latency is the sum of the times \( t_{pm}, t_{mb}, \) and \( t_{bb} \), our sub-goals are to minimize these terms. The distribution alternatives we describe vary the location of architectural components to highlight the affect of each alternative on these latency terms.

4.2 Distribution Alternative A

Alternative A is to run the patron applications, the manager modules, and the base applications on a patron’s computer (Figure 7). That is, a DL server only supplies base information. The patron opens base documents using appropriate base applica-
tions running on his or her computer. Marks and superimposed information are stored on the patron’s computer. Because all components run within the patron’s computer, the round-trip times between them would be quite low. Base documents would still be accessed over the network because they reside within the DL server. This cost is likely incurred only once per document per session, because documents accessed over the Internet are usually (automatically) first downloaded to the client’s computer. Base applications then access the documents locally. This alternative requires that each patron have all base applications locally, even for rarely encountered base types.

4.3 Distribution Alternative B

Alternative B is to run the patron applications on a patron’s computer, and run the manager modules and the base applications within a DL server (Figure 7). Marks are stored in the DL server, and superimposed information is stored on the patron’s computer. Because the base applications operate within the DL server, the patron is able to view base documents in their native applications only if those applications are also available locally on his or her computer. However, because the manager modules run inside the DL server, the mark activation operation would be unable to exploit any base application available on the patron’s computer. When the patron activates a mark (for example, the MS Word mark as in Figure 1 (b)), the DL server prepares the context elements needed to display the excerpt and sends it to the patron application (possibly in HTML as in Figure 4). The patron may request additional context elements to view as needed.

Fig. 7. Distribution Alternatives A and B. The dashed lines denote network boundaries

Because the manager modules and the base applications run within a DL server, the round-trip time between them would be low, but the round-trip time between a patron application and the manager modules ($t_{pm}$) would be high. Consequently, patron applications must strive to minimize the number of round-trips to the manager modules. For example, combining requests for context elements can reduce the number of round trips.
4.4 Distribution Alternative C

Alternative C is to run the patron applications and the two manager modules on a patron’s computer, but run the base applications within a DL server (Figure 8). As in Alternative A, marks and superimposed information are stored on the patron’s computer. Like Alternative B, the patron would be able to view base documents in their native applications only if those applications are also available locally. Unlike Alternative B, because the manager modules run on the patron’s computer, the mark activation operation would be able to exploit any base application available on the patron’s computer. In other cases, the Context Manager module would have to retrieve the necessary context elements to provide a view of marked regions. Because the manager modules run on the patron’s computer, but the base applications run within the DL server, the round-trip time between them \( t_{mb} \) would be high.

Fig. 8. Distribution Alternatives C and D. The dashed lines denote network boundaries

4.5 Distribution Alternative D

Alternative D is to run the patron applications on a patron’s computer, the two manager modules in a middle tier, and run the base applications within a DL server (Figure 8). As in Alternative A, the superimposed information is stored on the patron’s computer, but marks are stored in the middle tier. The capabilities of a patron application are similar to those in Alternative B. The performance of this alternative is also similar to that of Alternative B, except the round-trip time \( t_{mb} \) would be higher because the manager modules run in the middle tier. Finally, with the manager modules running in a middle tier, they can connect to more than one DL server.

4.6 Distribution Alternative E

Alternative E is similar to Alternative D except that the two manager modules and the base applications run in a middle tier (Figure 9). The performance of this alternative
is also similar to that of Alternative D, except that the round-trip time $t_{mb}$ would be lower because the manager modules and the base applications run in the same tier. The base documents are accessed over the network because they reside within the DL server, but the base applications are in a middle tier. That is, this alternative is similar to Alternative A with respect to base applications accessing base documents. However, the cost of accessing base documents could be less than that in Alternative A, if the bandwidth between the middle-tier and the DL server is better than that between the patron’s computer and the DL server.

As with Alternative D, the manager modules can connect to more than one DL server. Further, the same installation of base applications may also work with information on more than one DL server.

Fig. 9. Distribution Alternative E. The dashed lines denote network boundaries

4.7 Summary of Distribution Alternatives

Table 2 provides a summary of the location of components, the role of the DL server, and the profile of the components on the patron’s computer for each alternative. A DL server operating as an information server is similar to a file server, whereas an application server runs applications on behalf of clients. A thin client profile means a minimal amount of code runs on the patron’s computer. Patron applications tend to be browser-based (applets for example) in this case. A fat client profile means large amounts of code run on the patron’s computer. Patron applications tend to be desktop applications in this case, and are often richer in functionality than browser-based applications.

Table 2. Summary of alternatives. Client profile is the profile of the components on the patron’s computer

<table>
<thead>
<tr>
<th>Alternative</th>
<th>Location of components</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Patron apps: Middle tier, Managers: Middle tier, Base apps: DL, DL Server Role: App server, Client profile: Thin</td>
</tr>
<tr>
<td>E</td>
<td>Patron apps: Middle tier, Managers: Middle tier, Base apps: Middle tier, DL Server Role: Info server, Client profile: Thin</td>
</tr>
</tbody>
</table>

Table 3 summarizes the trend we expect for maintenance cost and performance of the resulting systems from the alternatives. Two rules of thumb drive our expectation...
of maintenance cost. First, a thin client is less expensive to maintain than a fat client because fewer components run outside a DL server in a thin client scenario. For components that run outside the DL server, changes made to a component need to be propagated to all locations where that component is deployed.

Second, the load on a DL server increases as the number of components running within the server increases. The trend Medium for Alternatives C and D indicates that the load on the DL server would be greater than that for Alternative A, but less than that for Alternative B. (The manager modules run outside the DL server in Alternatives C and D.)

Two rules of thumb guide our expectation of round-trip times: 1) placing components “closer” to each other reduces the round-trip time between them; 2) the reduction is greater if the number of round-trips between them is large (especially when the components exchange large amounts of data).

Based on these rules of thumb alone, Table 3 indicates that Alternative E has the best potential performance. The high round-trip time between a patron application and the manager modules ($t_{pm}$) appears to be its only weakness.

Table 3. Summary of maintenance cost and performance. Columns $t_{pm}$ and $t_{mb}$ are round-trip times as defined in Section 4.1

<table>
<thead>
<tr>
<th>Alternative</th>
<th>Maintenance Cost</th>
<th>Round-trip time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Patron apps</td>
<td>Managers</td>
</tr>
<tr>
<td>A</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>B</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>C</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>D</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>E</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

5 Discussion

In reality, DL systems are likely to employ a mixture of distribution alternatives. For example, some DL providers might wish to support patron applications of different capabilities (for example, fat clients and thin clients). Doing so requires the DL server to provide many kinds of interfaces (expose manager modules and base applications) for patron applications to choose from. Also, a patron may work with more than one DL, and those DLs may employ different distribution alternatives. In this case, patron applications must be able to discover the alternative a DL system employs.

Needs of patrons and patron applications, and access patterns also influence the choice of a distribution alternative. For example, Alternatives B and D are similar to each other except that the latter uses a middle tier. Though Alternative B causes a greater load on the DL server (see Table 3), its round-trip time between the manager modules and base applications ($t_{mb}$) is lower. If an analysis of the access pattern shows that the number of interactions between the manager modules and base applications is low, then the round-trip time $t_{mb}$ might not be critical. Also, Alternative B could be less expensive than Alternative D because the middle-tier would be eliminated from the configuration. Unfortunately, choosing an alternative is rarely this
easy, because the needs of patrons and patron applications vary (from time to time, patron to patron, and application to application). Fortunately, we can exploit some patterns to build flexibility into the system. For example, Alternatives B, D, and E have the common pattern that the manager modules are not resident on the patron’s computer. That is, patron applications cross network boundaries to connect to the manager modules in any of these alternatives, and the location of components beyond the manager modules should not matter to patron applications. Consequently, one could start with any of these three alternatives and migrate to another alternative as needs and access patterns change. The switching could even be transparent to patrons and patron applications.

The presence of a middle tier gives Alternatives D and E some advantages over the other alternatives. The infrastructure of the middle tier can be designed with DL services in mind. Also, there are likely to be far fewer middle-tier servers than patron computers. These alternatives also create the possibility for a federation of DLs to maintain a middle tier (or tiers), thereby distributing some of the cost of operations.

Distribution alternatives that deploy the manager modules and base applications outside a patron’s computer could also be helpful in serving requests from patron applications running on diverse operating systems. For example, SPARCE is currently implemented on the MS Windows® operating systems. This implementation could potentially be accessed using a web service to retrieve context for use in a patron application running on a Macintosh or UNIX operating system. (We are currently evaluating such a system of providing superimposed information services.)

Hosting base applications outside a patron’s computer may cause some problems. Without the necessary base application available locally, a patron will be unable to see a marked region in its original context (as in Figure 1(b)). In such cases, the context manager would have to retrieve the context elements necessary to provide a “broad enough” view of the selection, but the combined size of the context elements could be excessively large. Alternatively, the context manager could retrieve the context elements needed to display just the excerpt of the mark (but nothing surrounding it). In either case, the context manager needs to transform context elements to a format such as HTML or GIF, so the patron application may render the view. This transformation may not be easy for some base types. This problem is more likely to occur with Alternatives B, D, and E, because the manager modules run outside the patron’s computer, and they are unable to “call back” base applications on a patron’s computer. Mechanisms do exist for manager modules to call back applications on patron’s computers, but they present some concerns (for example, security). Also, calling back may require the manager modules to cope with many versions of the same base application across patrons’ computers.

Commonality and licensing are important issues related to base applications. Some base applications, such as an HTML browser, may be expected on all patrons’ computers, whereas an application such as Adobe FrameMaker® is unlikely to be available on all patrons’ computers. Some base-application vendors may disallow patrons from using installations resident on a DL server (or may require large license fees).

We have thus far discussed distribution and sharing of only executable code, but not the distribution and sharing of data. Sharing annotations is an emerging need among DL patrons. When sharing superimposed information, the corresponding
marks may be shared or replicated. It is also possible to share just the marks, but not the superimposed information that uses them. In reality, we envision that some marks and superimposed information may be shared, and some marks may be replicated.

Distributing components and sharing information could each increase security risks. DL systems may need to implement more than one alternative to balance security and performance. A small number of DL server interface points, and narrow functionality of those interface points can help reduce risk. The number of interface points in the SPARCE manager modules is far fewer than that in most base applications. They are also narrower in functionality. For example, The Mark Manager module has fewer than twenty methods, whereas the MS Word 10.0 type library has more than 30 top-level objects (each object has many methods) relevant to this discussion [7]. The MS Word Range object alone has 15 methods that return a “primitive” value and 21 methods that return complex objects or collections. It may be better to present an interface to the manager modules rather than to base applications if a patron connects to a DL server over a public network such as the Internet, but base applications may be exposed to a patron connecting over a trusted intranet.

In discussing the distribution alternatives, we mentioned that combining requests to retrieve context can help reduce latency. Such intelligence may be added to the Context Manager, thus benefiting all patron applications. Caching context may also be useful for some applications. Cached context could be used to minimize the number of round trips to a base application. Cached context could be useful when a base application or base document is inaccessible (ignoring issues of cache consistency). The location of the cached context may be chosen based on needs. A context cache placed within a DL server can help with requests from many patrons, whereas a cache on a patron’s computer can help with requests from only that patron. Alternatives D and E provide an excellent location (the middle tier) for such a cache.

Caching, replication and pre-fetching of base documents can also reduce latency in some cases. Caching could also enable offline access to base documents. The location of a cached document (or a replicated or pre-fetched base document) has some unique constraints compared to those for location of cached context. Unlike cached context, cached documents need to be accessible to base applications if a patron wishes to navigate to the document or retrieve context from it. Consequently, the document cache must reside on the patron’s computer in Alternative A. A document cache would probably not help in Alternatives B, C, and D because the base applications and base documents are collocated (on the DL server). However, Alternative E provides an excellent location (the middle tier) for a document cache. In general, a document cache could help when the DL server functions as an information server.

DL-server load balancing is another aspect that deserves consideration, especially in Alternatives B, C, and D. (It deserves greater consideration in Alternative B because both manager modules and the base applications run inside a DL server.) A DL server may replicate instances of the manager modules and the base applications to handle large number of requests. Replicated modules can serve concurrent patron

---

1 Other reasons, such as offline access by locally installed base applications, may motivate caching base documents on a patron’s computer even when they are inaccessible to base applications controlled by the manager modules.
requests, but in practice there is an upper limit to the concurrency obtained. The stress on system resources (for example, available main memory) will eventually prevent further replication, or even slow down existing replicas.

Dynamically choosing a distribution alternative provides an interesting means to reduce load on a DL server. For example, when using Alternative C, the manager modules could instantiate a base application on a patron’s computer if it is locally available (thus switching to Alternative A), and use base applications on the DL server in other cases. Alternatives D and E could be mixed similarly (in the middle tier). Distribution architectures could also be different for different base types.

6 Related Work

The DELOS-NSF Working Group’s report on Digital Library Information-Technology Infrastructures [1] highlights the importance of services and infrastructure for metadata and annotation services in DLs. OAI-PMH [13] and its extension XOAI-PMH [12] have demonstrated the feasibility of component-based architectures for metadata and annotation services respectively in a DL.

The UC Berkeley Digital Library Project uses superimposed behaviors in multivalent documents to support annotations [15]. That work facilitates distributed annotation and base data, but not distribution of executables. The Stanford InfoBus [9] defines a mechanism for interaction among UI clients, proxies, and repositories. (SPARCE’s manager modules may be viewed as proxies.) It also defines service layers that are available to clients, proxies, and repositories. However, it does not consider distribution of executables.

FEDORA [10] and XOAI-PMH [12] provide promising frameworks for integration of superimposed information services with other DL services. The parameterized disseminators of FEDORA could be used to address (access) parts of documents. XOAI-PMH does not explicitly specify sub-document objects, but its extensibility mechanism could be used to create item instances that serve a similar purpose.

Caching, replication, and pre-fetching of DL documents have been researched more extensively [2, 3, 4] than caching of context information. Past work on caching metadata in hierarchical harvesting systems [5] provides useful insight into issues related to caching context information.

7 Summary

We have described five distribution alternatives to provide superimposed information services in DLs and defined some metrics to compare the performance of the alternatives. We have illustrated the distribution alternatives using SPARCE, our middleware architecture for superimposed information management. We have also discussed some of the advantages and disadvantages of employing the distribution alternatives.
References