ELECTRONIC PUBLISHING, VOL. 7(1), 35–48 (MARCH 1994)

Distributed Documents: an architecture for open distributed hypertext

A. HATZIMANIKATIS AND I. GAVIOTIS
Department of Computer Engineering and Informatics
University of Patras
Greece

D. CHRISTODOULAKIS
Computer Technology Institute
Patras
Greece

SUMMARY

A conceptual design for our architecture centered around the entities of a hypermedia node, link, anchor and document is initially presented. Each entity has a well-defined interface so that the respective instances can cooperate despite the number of different media types. Virtual documents are created as views on other documents borrowing from their content and customizing their behavior during navigation and editing.

The system functionality is provided by hypertext document objects, acting as providers of hypermedia services. There are storage and display services which are accessible and consumable by the local and remote clients spanning the operating system and workstation boundaries. Due to the object-based approach taken at design and implementation, the incorporation of new types of services (general and media specific) is straightforward and integrates smoothly with the rest of the system.

KEY WORDS Hypermedia systems Hypermedia services Distributed systems Open architectures

1 INTRODUCTION

The explosion in the quantity of available on-line information has, paradoxically, obviated the need for elaborate document handling systems. Recent advances in the fields of information retrieval, expert systems and multimedia databases address this issue by attempting to equip information workers with new, or enhanced, searching and authoring capabilities. The hypertext approach has gained support in this respect because, not only can it structure a large volume of data, but also it has proven adequate in supporting the diversity of types and media that are used for representing this information.

The first and second generations of hypertext systems were stand-alone applications which organized their data using hypertext principles for interconnections between the data items; in this way they could offer facilities for navigation and management of the hyperspace. It was soon realized that, helpful as these applications could be, they created, in effect, isolated 'hyper-islands' of documents whose borders could not easily be crossed by the users of the applications.

For this reason, a number of recently developed hypertext and hypermedia systems avoid constraining the user to a single application, but rather they choose to cooperate with a (closed) set of applications. This approach is exemplified by Intermedia [1]. Intermedia supports a set of cooperating applications integrated in a hypermedia framework. Links

CCC 0894–3982/94/010035–14

Received September 1992
Revised October 1993

can be created and managed between different media types and the notion of *hypermedia documents* is supported.

The architecture proposed here, Distributed Documents or D^2, elaborates on the *openness* aspect by allowing easy integration of new applications and new media types. This kind of flexibility is not just restricted to new, custom-built, hypertext-aware applications; it also extends to "uncooperative applications", which cannot be modified to include the hypertext functionality directly. Sun's Link Service [2] follows a similar approach, but it requires the applications using the service to have explicit knowledge of hypertext functionality.

One further step in the direction of implementing an open system is to cater for the *distribution* of application services as well as the hypermedia documents themselves. D^2 does not confine the user to a functionally closed system; moreover it exploits the network to provide access to remote information and to the programs which handle that information. As a general rule, information will be produced, revised and accessed on many computing sites, so it is natural to extend the hyperlinking model so that it supports navigation inside the computer network. Knowledge workers—our target group—require transparent network movement together with easy, efficient manipulation of network resources.

Apart from *openness* and *distribution*, one of our goals for the D^2 architecture was that it should be able to support *hypertext views*, i.e. different hypertext organizations on the same underlying information. We regard personalized views as a crucial feature, when dealing with large information spaces.

The principle of separating the hypertext functionality from the storage and the user interface functionality has already been recognized by many researchers in the area of hypertext [3–5]. We have chosen to follow this principle because it provides the necessary flexibility for dealing with complex, heterogeneous, distributed environments. Different storage formats and capabilities, and different user-interface paradigms and functionality, coexist in such environments, and the hypertext architecture must be ready to adapt to them. In this work, we concentrate on the hypertext functionality and we describe a model that provides a set of *generic hypermedia services* integrated into a distributed environment. The Distributed Documents architecture is an effort to structure this functionality in order to support the kind of services, and to achieve the kind of goals, that we have mentioned above.

In the next section we present the Distributed Document architecture, the hypertext data model it supports, and the associated functionality. In Section 3 we describe how this functionality can be implemented in a distributed environment and we present HyperPress, a prototype application that was developed based on the architecture described. A comparison with related systems is given in Section 4, and the last section sets out some conclusions.

2 THE DISTRIBUTED DOCUMENTS ARCHITECTURE

The design of our architecture is based on the concept of *service* provision and consumption. A service can be accessed through a number of *operations* that are associated with it. Operations constitute the *interface* of the service. Services are produced and consumed by *objects*. Objects can be dynamically created and destroyed. The Distributed Documents architecture consists of a number of cooperating objects which provide a set of hypertext services to the rest of the system.

In the next subsection we present the hypertext data model, supported by this architecture and we describe the services associated with each entity.

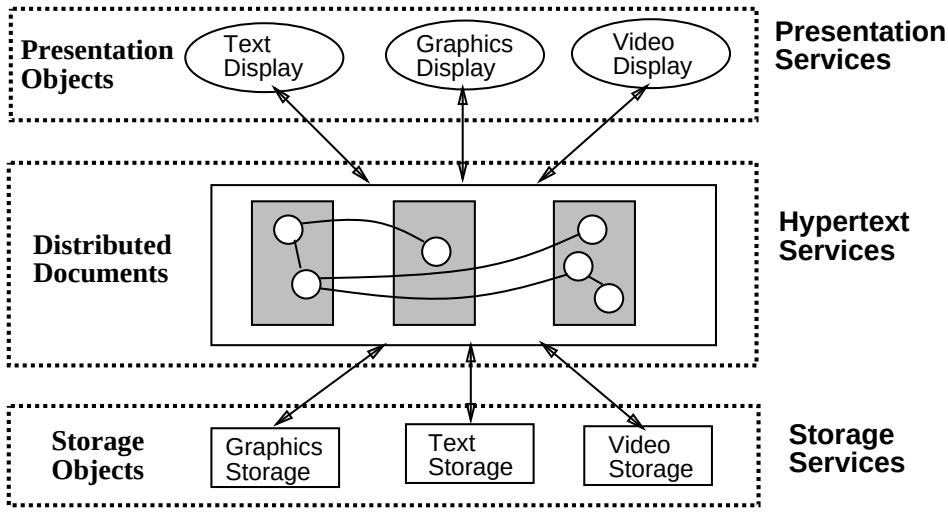

Figure 1. The entities of the architecture

2.1 The data model of the Distributed Documents architecture

Our architecture supports a simple and general hypertext model. In Figure 1, we show the entities and relations that make up the Distributed Documents architecture in the form of an entity-relationship diagram. The basic entities are *nodes*, *links*, *anchors* and *hypertext documents*; a detailed description of each of these, and their structural relationships, now follows.

The *document* is the most important entity in the architecture. We define a document as the packaging of a set of nodes and a set of links. A document can consist entirely of links and have no nodes at all—this is very useful in the case where a user wishes to add his own set of relations between nodes that exist in different documents.

Nodes can be *simple* or *composite*. A simple node contains a chunk of *media*, e.g. text, image, or video, while a composite node contains several media chunks, which are meant to be manipulated or displayed together as a single entity. The same chunk can belong to more than one node, so that different hypertext structures can be imposed on the underlying information. Furthermore, *display information* is associated with each composite node. This information takes the form of synchronization or layout constraints, and describes how the various media chunks, constituting the composite node, should be presented to users. The same display information can be associated with more than one composite node and may be common for all composite nodes belonging in a document. A node cannot belong to more than one document, with the exception of *view* documents, described later. Each node, simple or composite, contains a set of *anchors* and each anchor belongs to exactly one node.

The *anchor* abstraction plays a central role in the architecture. Anchors describe *media areas*, spatial or temporal, referring to a media chunk. An area could be the third paragraph of a text chunk, or a time frame from the third to the fifth second in a video sequence. Apart from the user interface, the description and the nature of the media area is not relevant to any other part of the system that is used for displaying anchors to users. Anchors encapsulate

this area information, so that they are treated uniformly throughout the system and are accessed through a single identifier. A node contains a set of anchors associated with the media chunk(s) it contains, and these form the source or destination points of *links*.

Links connect a *source* anchor to one or more *destination* anchors. They have no knowledge of the media areas encapsulated in anchors. Even when the area in an anchor is changed, e.g. the string associated with a text anchor is edited, the link still maintains the correct information. Each link belongs to only one document, but the anchors it associates can belong to different documents which do not necessarily correspond to the document that the link belongs to.

View documents are *virtual* documents, in the sense that parts of them are dynamically computed. A view document may contain a set of links and nodes, but it also contains a *computation*, in the form of a script or a query to other documents. When the computation is executed the contents of the view are updated with the results of the query. These results are just references to nodes and links that belong to other documents. Documents already impose different organizations on the underlying stored information. Using views, a second level of structuring is introduced.

There are some comments that should now be made about the model we have just described. First, we make the assumption that each instance of the entities described can be uniquely identified across the system. Second, a *property list*, a list of *attribute–value* pairs can be associated with each entity instance. Queries on attributes, such as date or author, are the simplest items that can be used for the creation of view documents.

For each entity described, there is a corresponding *object class*. Thus, we find a *document class*, *node class*, *link class* and *anchor class*, that provide the corresponding services. These classes, and the services they provide, actually constitute the Distributed Documents architecture, and are the focus of our interest. In the next subsection their functionality is presented in detail, together with the storage and presentation services they use.

2.2 Functionality of the components

As we have already mentioned, the hypertext layer consists of a number of cooperating objects that provide the hypertext services. In the same way, both the storage layer and the user-interface layer can be modeled as a set of services provided to the hypertext layer (Figure 2).

The storage layer consists of a number of *storage* objects, which can be database managers, object-oriented databases or even plain file systems; these provide the storage and retrieval services for media chunks. Special storage objects can be used for the storage of different media, according to the particular media requirements.

The topmost, user-interface, layer consists of *presentation* objects, sometimes called *readers* [3] or *media editors* [6]. They provide the service of rendering a node and presenting it to the user for viewing or editing. Presentation objects support an operation which takes, as input, an identifier for the chunk to be presented, together with the identifiers for the corresponding anchors. The presentation objects then cooperate with the storage objects to retrieve the media chunks and, depending on their degree of integration with the hypertext layer, they should also be able to present anchors and links and to interpret display information associated with composite nodes. For example, a text presentation service should provide an operation that takes a text anchor (i.e. a text area specification) as input and then highlights the corresponding text. There are also *document presentation*

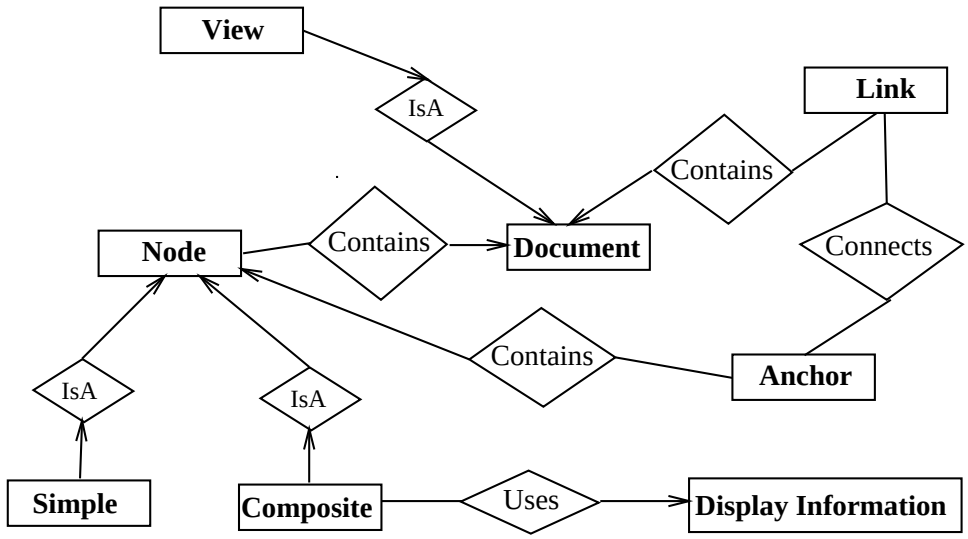

Figure 2. Architecture and system services

objects, which are responsible for displaying a whole document in the form of a graphical map or a summary of its contents.

In addition to the sets of nodes and links that we have already discussed, each document object also has to provide the service of maintaining the hypertext graph. For example, it will accept requests for the addition and removal of nodes and links, for displaying its content on a screen or for a version of the document to be saved into a file. These operations constitute the interface of the hypertext document service. Note that the document can impose constraints on the addition and removal of nodes and links, as well as constraints on browsing and navigation. Some documents might maintain specific versions of nodes, while others would not allow editing; or, again, some might accept queries on their contents or structure, so that view documents can be created out of them. It should be noted that all documents provide the same *document* service to the rest of the system (i.e. they support the same set of operations) but the semantics of the operations may well be different for different kinds of documents.

The main service provided by the node object is that of communicating and presenting its contents and anchors to a corresponding presentation object. A node object uses the services of the storage layer for maintaining its contents and the user-interface layer for presenting its contents. Anchors are responsible for updating themselves when a user edits the contents of the corresponding node, for example when the string associated with a text anchor is edited.

Links are required to provide services for navigation and for displaying themselves. When a link is navigated, it is responsible for displaying its destination nodes, by using the display operation provided by the node object. Special action is taken whenever a link is navigated that crosses document boundaries so that users can be made aware that this action has taken place.

Our process model is similar to Arjuna's model [7]. A process is associated with each document object, and is responsible for servicing requests made to it. Inside this process,

multiple threads of control are created whenever operations on nodes and links are invoked. For example, a new thread is created when a node belonging to the document is displayed or when a link is navigated. Documents can be *passive* or *active.* A passive document is one that is stored and no process is yet associated with it. It becomes active after an explicit user request asking for navigation in the document, or when a hyperlink is followed that points to a node belonging to the document. A document may also become active if it receives a request to compute a query made by a view document.

View documents are treated in exactly the same way as other documents. They provide the same set of operations and the same services. The only exception is that when a view document is activated it makes requests to other documents for the execution of a query rather than using an internally stored hypertext graph. The results—references to nodes and links belonging to other documents—are merged and the view document is presented to the user in the same way as any other document. A view document can become permanent if a user explicitly requests it. In this case copies of the nodes and links are created, which take the place of references, and which are stored independently of the original nodes and links.

It should be clear by now that the implementation of the architecture provides a set of services to the rest of the system, consisting of a number of cooperating multithreaded processes belonging to particular documents. The notion of the document serves two main purposes. First of all, conceptually, it is used for grouping related information. Treating views as documents helps to keep the architecture simple and provides a uniform model to the user. Secondly, the encapsulation of documents in a single process and the abstracting the notion of a 'document' away from storage and presentation issues, makes for the simple implementation and distribution of services, as will be shown later.

In the remainder of this section we introduce the notion of computations in hypertext and try to justify our assertion that the architecture is both open and extensible.

2.2.1 Computations in hypertext

An issue that deserves our attention is the introduction of computations in the hypertext structure, i.e. ways for dynamically altering the structure of the hypertext graph. In the Distributed Documents architecture, computations can be introduced at different levels. First, at the document level, the view documents introduce computations that have new hypertext structures as their result. Second, at the link level, the *navigate* operation of a link can be implemented in such a way that it can choose one out of several possible destination anchors, depending on the user and the previous path taken through the document. Link objects are also responsible for activating a document whenever a hyperlink is navigated that crosses document boundaries. Indeed, it is even possible for different documents to be activated, depending on the result of a computation performed by the link. Finally, computations can be introduced at the anchor level. The act of arriving at an anchor, after navigating a link, can result in the activation of another application, or the execution of a piece of code encapsulated in the anchor.

2.2.2 Adding new documents

It has already been mentioned that different documents can exhibit different behavior and semantics with respect to the maintenance of the hypertext graph, and the facilities for

navigation and browsing. If a user wants to create a new kind of document, he has to create a *document template*. This is the code for a process that provides the same operations as for all other documents, but possibly with different semantics. Document templates can be instantiated, and links and nodes added, so that instances of the new kind of documents are created. It should be noted that already-existing nodes and links, and their operations, can readily be used by the new document templates. The system is open, in the sense that the new documents can cooperate with existing ones without any modification, and links can be created between diverse types of document.

2.2.3 Adding new media

When new media are going to be added, the user should specify the corresponding node and anchor objects, and make available the corresponding storage and presentation objects. Already-existing documents can link to nodes containing the new media, as long as the new nodes support the same operations as the already existing ones. Using the abstraction of node and anchor the architecture is extensible in the sense that it can readily integrate new media.

2.2.4 Integration of already-existing applications

The architecture supports a number of different ways for integrating already existing applications, such as text or graphics editors, into the environment. The simplest possibility is for these to be integrated at the level of presentation services. Thus, a conventional editor might be used as the presentation service for text media. The problem is that these editors will not be aware of anchors; we have to accept that the anchor is the whole text, or find a separate way of displaying anchors. Finally, applications such as databases or indexing systems can be encapsulated in anchors. The result of displaying such an anchor is the invocation of a query to the application and the presentation of the results.

The description of the architecture, as presented thus far, can be easily implemented in a computing environment supporting multitasking, e.g. UNIX. In the next section we show how this architecture can be transferred in a distributed environment.

3 DISTRIBUTION ASPECTS

One of the first requirements for the Distributed Documents architecture, as implied by the name chosen, was that it should be easy to implement in a distributed environment, i.e. an environment consisting of heterogeneous workstations connected using a network. When working with distributed applications it is generally the case that processing and data are transparently partitioned across the network, in order to achieve, among other things, better performance and higher availability. Workstations on the network may have different architectures and capabilities, so the system should be able to cope with heterogeneity.

Most of the distributed hypertext systems that have been implemented until now, are based on the idea of a single server maintaining the hypertext graph, while a number of clients, running on different workstations, access the database [8,9]. The architecture we propose allocates parts of the hypertext graph, and the processing associated with it, to different workstations.

We begin with the realistic assumption that workstations on the network support a variety of storage and presentation services. For example, a workstation may possess mass storage devices, ideal for the storage of continuous media such as video or sound. Another workstation may have limited storage, or none at all, but may be equipped with I/O devices for presenting video and sound. There are constraints on the services a workstation can provide and on the functions it may perform.

The main idea in distributing the architecture is that a *document* is the unit of distribution. This means that a document can be activated, or made passive, on any workstation across the network. The process associated with the document can fail, or recover, independently of the others. So, apart from storage and presentation services, there are *hypertext document services* distributed across the network. Document servers are responsible for the maintenance of their own hypertext structure; they are clients of the presentation and storage services whenever they wish to present or store the contents of their nodes. Furthermore, they are clients of other document servers, so that hyperlinks between different documents can be created and navigated and view documents can be computed.

There are a number of issues that arise in a distributed environment. In the following, we deal with some of them and give hints for an implementation. It should be noted, that we have implemented a first prototype of the proposed architecture based on ANSAware [10].

ANSAware is a platform for the development of open distributed applications in a heterogeneous environment. It provides lightweight threads, remote procedure calls and distributed naming over different operating systems. Furthermore, it supports a model for persistent objects in which these objects can be passivated and activated dynamically [11]. The platform provides the application programmer with a *stub compiler* and a preprocessor for a *distributed programming language*, that can be embedded in C code. The stubs created care for the marshaling and unmarshaling of arguments passed between a server and the clients of a service. Moreover, there is a name server available, called *trader* [12]. Name services are used in order to provide location transparency in a distributed system. In our case we have used the trader in a slightly different way. The trader is actually a directory of the available services in a system. It stores the types of the services, i.e. the interfaces they support, as well as references to the servers that provide the service. Each server is associated in the trader with a number of programmer- or user-defined properties. The binding of clients to services is done at run-time. When a client wants to use a service, it queries the trader with the type of service needed and the properties required of that service. Services are organized in a hierarchy, so that the operations supported by a child service are a superset of those supported by the parent.

In our case (Figure 3), there are document, storage and presentation services. The document services are characterized by *author* and *title* properties. The storage services are divided into services for text, images etc. and a similar division can be found in the presentation services, which are characterized by the host they are running on. When a document is active and is asked to display the contents of a node, the trader is asked if there is a suitable presentation service available on this node. By contrast, when the contents are to be stored, the trader is asked for a suitable storage service. If more than one service of the same type is available, then properties such as the cost of using the service, and its performance, are evaluated in the course of determining the best service to use. The trading service is very powerful, and it is heavily used in our architecture in order to achieve the required flexibility.

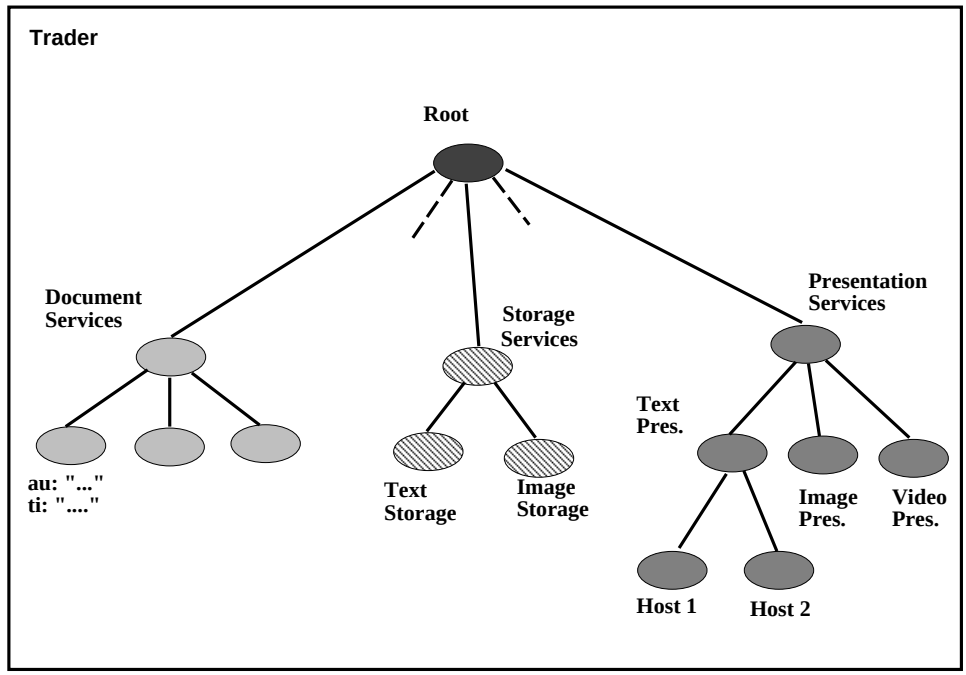

Figure 3. Use of the trader in the Distributed Documents architecture

3.0.1 Document activation

When a document is activated on a workstation a new process is associated with it and the hypertext graph associated with the document is retrieved from storage. It is possible that the document is activated on a workstation different from the one on which it was stored (remote activation). When this happens the hypertext graph is cached to this workstation. We expect that the quantity of cached data will be fairly small, given that the contents of the nodes are stored separately. Furthermore, while a user is viewing the contents of a node, the contents of the neighboring nodes are cached; in this way the navigation of a link can be greatly speeded up. Images have priority for cacheing, on account of their large size, and nodes containing video can try to allocate the network bandwidth required for their display [13]. In the case of remote activation there is the possibility that processes associated with the same document can exist on different workstations. This creates problems for the integrity of the original document. We expect that a single document will normally have permissions such that it can be edited by one user at a time; other users will have read privileges only. If this is not the case, we choose an *optimistic* scheme: we presume that any conflicting changes can be resolved when the document is stored back to the workstation from which the passive document originated. This is much less complex than having to inform all the associated processes about the changes to the document as they are being made. Different kinds of documents can adopt different techniques for resolving conflicts, such as creating new versions, or trying to merge changes, or even informing the user of the conflicts so that special actions can be taken.

3.0.2 View computation

Having the documents as processes on different workstations allows us to execute queries on documents in parallel. When a view document is activated, requests are sent to all relevant documents over the network and are computed in parallel on different workstations. The results, usually small parts of the documents, are returned to the originating view process.

3.0.3 "Publishing" a document

Some users may not wish to present their documents to other users of the system. This is usually the case with views containing personal annotations. When users want to make a document available, i.e. to 'publish' it, they can register it in a name service. In our prototype implementation we haved used ANSAware's trader for the registration of "published", widely available documents, as we have mentioned earlier.

It is true that this distributed architecture encourages small- to medium-size documents, as it is not efficient to cache even the structure of big hypertext documents. But it is also the case that big hypertext documents pose problems to users, in term of orientation and cognitive overhead. It is our belief that small, well-structured documents should be better supported and encouraged by the underlying hypertext architecture. Good and efficient management of long and complex hypertext documents could be achieved through the concept of views.

3.1 A case study: HyperPress

We have implemented a prototype of a distributed hypertext system, based on the Distributed Document architecture. The prototype was built with specific, realistic, requirements in mind, namely the requirements of a press agency. In such an environment there are a number of information sources. There are news pieces coming from different news agencies and there are news pieces written by the journalists. We wanted to have everything integrated under a single hypertext system, distributed over a network of workstations.

The information coming from the different sources is stored and different hypertext documents are created from it. Each piece is associated with a keyword, e.g. international affairs, sports or culture, a date, an author and a city, where the events took place and where the correspondence came from. The documents are created based on the date, the keyword and the source agency. There is a number of automatic linking services, actually different processes, that take as input a news piece and try to create links to news pieces that already exist, based on probabilistic methods for the characterization of text [14]. When the links are created, they are added to the corresponding document. This facility is experimental, but, so far, it has given quite good results. It has certainly been a good test for our architecture, since it demonstrates how different services can be integrated into a single environment. The automatic linking facility uses the same document services that are available to any user who adds nodes and links by hand. Indeed, not all of the links are automated by this facility. There are links that are created manually, by special users, and some of the links are explicitly placed in those news pieces that are updates of previous ones.

Views are created on the documents, based on the name of the author, the city where the

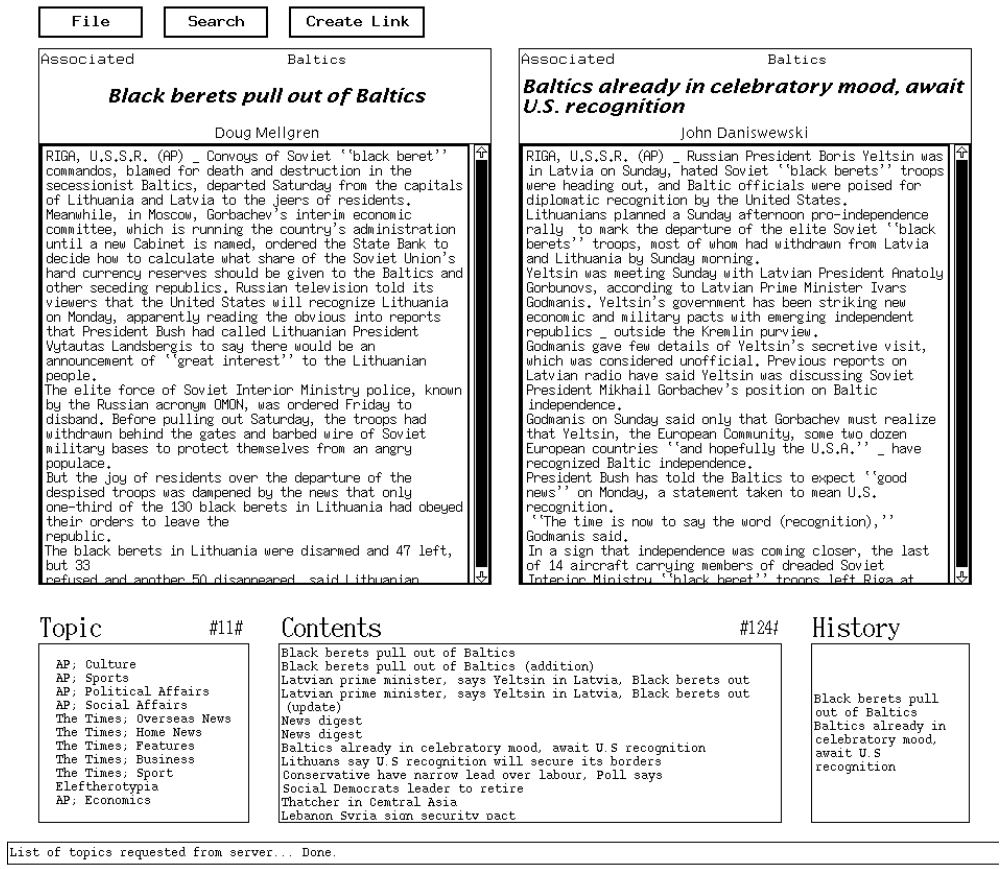

Figure 4. The user-interface of HyperPress

correspondence comes from, or both. This way users have the option of tracing the whole history of an event. Furthermore, journalists possess their own documents and views, where they add annotations or gather news pieces that are of interest to them.

The system supports text and still images, i.e. photographs that come in from agencies. Usually, a photograph is associated with a caption, and a composite node is created out of them. There are special storage servers, that handle large amounts of textual information, and a separate storage server for images. There are presentation servers for the text and images running under X Windows on Unix workstations. In Figure 4 the user-interface of the prototype HyperPress application is shown. Users can choose the topic they are interested in, and then a specific article from the contents. They can navigate from article to article, while the last two articles visited are always shown. A history of visited articles is always available. Presentation services for MS Windows are under development.

The architecture has provided us with a simple and powerful model for handling the problem of the various information sources and access requirements. A document template has been created for news pieces coming in from the agencies, another one for the creation of views and a third for user documents. Later, Internet news was integrated into the system, using the same document templates. Each document belongs to a single user, who

has “write” access for it, while other users have “read” privileges only.

The performance of the system is satisfactory, but work continues on improving it, mainly in the areas of further experimentation with the automatic linking facility and the construction of improved presentation services for a range of workstations and printers. User requirements are still being investigated, so that better view documents can be developed.

4 OTHER RELEVANT WORK

Recently, many hypertext experts strongly advocate the transfer of hypertext functionality from the application level to the operating system level. Two examples of this approach are Microsoft’s Object Linking and Embedding (OLE) and HP’s New Wave, which are supported by some PC-based applications. Following this trend, architectures have been proposed which enable the integration or better overlaying of hypertext functionality, just above the operating system level. One notable example is IRIS Hypermedia Services [15]. In Intermedia, there has been an effort to separate the general hypertext functionality from the front-end applications. To this end, all information related to links and anchors is stored in the document server which is reached via interprocess communication from the rest of the system.

Our architecture, D^2, has elaborated the issue of separation between applications and the hypermedia substrate. From the previous description, one observes that D^2 is open to new applications whose files are readily incorporated into hyper-documents. Moreover, by use of the storage and presentation servers, we have attained remarkable independence from the file system and the display equipment. Our assumptions concerning the idiosyncracies of the underlying platform are kept to a minimum.

A document supported by D^2 is a collection of multimedia hypernodes. A single hypernode can itself be composed from many media types such as in [3]; in this case, the navigation of a link results in the activation of many applications displaying the parts of this node. We think this navigation semantics is more natural than restricting the destination of a link to be unimedia.

In contrast to IRIS, each document holds its own collection of links, anchors and nodes and it does not rely on a single central database server for the provision of access and querying facilities. The distribution granularity is more fine-grained with the potential for greater concurrency and document availability. Moreover, D^2 adopts, universally, a client/server organization: the hypertext functionality depends on presentation servers to display and revise the content of the document, and on storage servers to access and make use of non-volatile storage devices, etc. These servers can be reached from the trading service and supplied to any interested process. The provision of clear interfaces between processes alleviates the tasks of the application programmer because each object’s role is unambiguous.

D^2 is general. Its implementation based on the ANSAware distributed platform makes it operable on heterogeneous systems. As development of media-specific and equipment-specific servers progresses, hyper-documents will be accessible by a multitude of machines. Our approach of proxy servers is a powerful and elegant solution to the problem of ‘un-cooperative applications’. In [4], proxy objects are implemented in a restricted way: an application which does not use the messages of the hypertext protocol can be activated to execute a particular action defined as a script in a proxy anchor object. When this ‘foreign, guest’ application takes control, the hypertext layer becomes inaccessible and any navi-

gation facilities cease. On the contrary, the D^2 approach is based on encapsulation of the 'foreign' applications with a server interface which is communicating with the hypertext system. This server takes the control when the client application concludes and the session resumes without any loss of continuity.

5 CONCLUSIONS AND FURTHER WORK

The Distributed Document architecture is an example of the feasibility and merits of open distributed hypertext. It uses a simple conceptual model, that of the hypertext document, for the packaging of hypertext structure and the provision of hypertext services. These services can be provided transparently across a distributed environment. The architecture has proved its validity during the construction of the HyperPress application.

Our work is currently concentrating on the integration of diverse applications into the environment and the development of corresponding document templates. Furthermore, we want to investigate the integration of hypermedia document-exchange formats, such as HyTime [16], as the storage formats for documents. Through this process, we expect some further refinement of the architecture as well as the discovery of its flaws and limitations.

REFERENCES

1. Nikole Yankelovich, B. Haan, Norman Meyrowitz and S. Drucker, 'Intermedia: The concept and the construction of a seamless information environment', *IEEE Computer*, **21**(1), 81–96, (January 1988).
2. Amy Pearl, 'Sun's link service: a protocol for open linking', in *Hypertext '89 Proceedings*, ACM, November, 1989, pp. 137–146.
3. Frank W. Tompa, 'A data model for flexible hypertext database systems', *ACM Transactions on Information Systems*, **7**(1), 85–100, (January 1989).
4. Charles Kacmar and John Leggett, 'PROXHY: a process-oriented extensible hypertext architecture', *ACM Transactions on Information Systems*, **9**(4), 399–419, (October 1991).
5. Frank Halasz and Mayer Schwartz, 'The Dexter hypertext reference model', in *Proceedings of the Hypertext Standarization Workshop*, Gaithersburg, MD, NIST, January 1990, pp. 95–133.
6. George D. Drapeau and Howard Greenfield, 'MAEstro—a distributed multimedia authoring environment', in *Multimedia—For Now and the Future,Proceedings of the Summer 1991 Usenix Conference*. The USENIX Association, 1991, pp. 315–328.
7. Santosh K. Shrivastava, Graeme N. Dixon, and Graham D. Parrington, 'An overview of the Arjuna distributed programming system', *IEEE Computer*, **8**(1), 66–73, (January 1991).
8. B. Campbell and J. Goodman, 'HAM: a general-purpose hypertext abstract machine', *Communications of the ACM*, **31**(7), 856–861, (July 1988).
9. I. Gaviotis, A. Hatzimanikatis, and D. Christodoulakis, 'An architecture for active hypertext on distributed systems', in *Proceedings of the Sixth Annual European Conference on Computer Systems and Software Engineering*, The Netherlands, IEEE, May, 1992, pp. 377–382.
10. Architecture Projects Management Ltd., Poseidon House, Castle Park, Cambridge, England, *ANSAware 4.0 Document Set, Vol. A–D*, March, 1992.
11. Michael H. Olsen, 'A prototype of the ANSA storage model', ISA Project RC.364, APM Ltd, Poseidon House, Castle Park, Cambridge, England, June, 1992.
12. Jean Pierre Deschrevel and Andrew Watson, 'Trading service overview', Technical Report APM/RC.324.01, Architecture Projects Management Ltd, Poseidon House, Castle Park, Cambridge, UK, July, 1992.
13. Shoshana Loeb, 'Delivering interactive multimedia documents over networks', *IEEE Communications*, 52–59, (May 1992).

14. Gerard Salton, 'Automatic text indexing using complex identifiers', in *ACM Conference on Document Processing Systems*, ACM, December, 1988, pp. 135–144.
15. Bernard Haan, Paul Kahn, V.A. Riley, J.H. Coombs and N.K. Meyrowitz, 'IRIS hypermedia services', *Communications of the ACM*, **35**(1), 36–51, (January 1992).
16. Steven Newcomb, Neill Kipp and Victoria Newcomb, 'The HyTime Hypermedia/Time–based document structuring language', *Communications of the ACM*, **34**(11), 67–83, (November 1991).