Addressing Interoperability in Open Hypermedia: The Design of the Open Hypermedia Protocol

Sigi Reich[1], Uffe K. Wiil[2], Peter J. Nürnberg[3], Hugh C. Davis[1], Kaj Grønbæk[3], Kenneth M. Anderson[4], David E. Millard[1], Jörg M. Haake[5]

[1] Multimedia Research Group, University of Southampton, UK {sr, hcd, dem97r}@ecs.soton.ac.uk

[2] Dept. of Computer Science, Aalborg University Esbjerg, DK ukwiil@cs.aue.auc.dk

[3] Dept. of Computer Science, Aarhus University, DK {pnuern, kgronbak}@daimi.aau.dk

[4] Dept. of Computer Science, University of Colorado, Boulder kena@cs.colorado.edu, USA

[5] GMD-IPSI, Darmstadt, Germany haake@darmstadt.gmd.de

ABSTRACT: Early hypertext systems were monolithic and closed, but newer systems tend to be open, distributed, and support collaboration. While this development has resulted in increased openness and flexibility, integrating or adapting various different tools, such as content editors, viewers or even other link servers has remained a tedious task. Many developers were implementing essentially similar components, simply for the benefit of having their own platform on which to experiment with hypertexts.

The open hypermedia community is addressing this issue of interoperability between open hypermedia systems. The goal of the community is to provide an open hypermedia framework that can be used by application developers outside the community to construct more powerful hypermedia-aware applications. The design and evolution of this framework is presented along with the requirements that drove its development. The framework has matured to the point where it has supported the creation of a number of research prototypes. These prototypes are described and evaluated with respect to their use of the framework.

KEYWORDS: Open hypermedia protocol navigational interface (OHP-Nav), component-based open hypermedia system (CB-OHS), reference architecture, data model, interoperability, open hypermedia systems working group (OHSWG), structural computing.

1 HISTORY AND MOTIVATIONS

"Hypertext and hypermedia technologies have reached the point where it makes sense to consider their potential for formal standardization"(1). This statement is taken from the call for papers of the "NIST Standardization Workshop on Hypertext" which took place in January 1990. Hypermedia technology has become more pervasive since then. Several open hypermedia prototypes are in existence, including Chimera (2), DEVISE Hypermedia (DHM) (3), HOSS (4) and HyperDisco (5). Furthermore, there even exist commercial products which have evolved from research prototypes such as HyperWave (<http://www.hyperwave.com/>, (6)) and Microcosm (<http://www.multicosm.com/>, (7)), and 1994 saw the rapid growth of the World Wide Web (WWW). Thus, nearly a decade later, hypermedia technology is available to a much wider community of users.

However, the standardization process has lagged behind the development of new technologies. While the WWW has always based its development on community-derived extensible standards, open hypermedia developed independently in various research groups, without the benefit of a standard set of services and protocols agreed to by the open hypermedia community. This lack of a common understanding resulted in a collection of systems providing similar services that could not interoperate to any degree.

There are many beneficiaries to interoperable open hypermedia systems: end-users, content providers and developers. From an end-users' point of view for example, interoperable systems would allow the use of hypertext functionality in a standardized way, similar to features such as cut/copy/paste that are so common in today's systems (8). Furthermore, users would be better able to choose between different vendors of hypertext applications because basic functionality e.g., for navigation, would be supported by all systems. And finally, consumers would be able to re-use others hypertexts, i.e., the anchors, links, tours, trails, etc. that have been authored, often with a lot of effort, in a similar way that people exchange bookmarks today.

Content providers would benefit in that their products could be re-used on multiple platforms and systems. Hence, interoperable hypermedia systems would form the foundation that would allow information re-use. Not only would this decrease costs but also it would improve the usability and thus the quality of hypertexts.

Developers on the other hand benefit from standards by being able to re-use tools and components. So some might specialize in writing scalable high performance servers; others might specialize in implementing feature-packed clients. Also, integrations or adaptations would only have to be done once (9). Furthermore, the increased availability of standard tools would result in a proven and stable platform on which developers could prototype and evaluate their new tools.

In 1994, the open hypermedia community decided to address the lack of interoperability between open hypermedia systems at the first International Workshop on Open Hypermedia Systems (OHSs) (10). At the next workshop, the Open Hypermedia Systems Working Group (OHSWG) formed to coordinate the efforts of the community in creating standards that would address the interoperability issue. The OHSWG meets regularly at each ACM Hypertext conference (OHS 1 (11), OHS 2 (12), OHS 3 (13), OHS 4 (14) and OHS 5 (15)) and at working group meetings held in between each conference (OHS 2.5, OHS 3.5, OHS 4.5 and the upcoming OHS 5.5). These meetings serve as a forum for presenting research on open hypermedia systems and reporting progress on the standardization efforts. This paper reports on the evolution and technical details of the interoperability standards being created by the OHSWG. The issues covered by this work overlap the issues discussed at the NIST standardization workshop: a common data model, a reference architecture, interfaces and protocols, and many more. However, in contrast to that workshop, the OHSWG has reached agreement on many standardization issues and can begin to publish the results of its work. Underscoring this achievement, members of the OHSWG have produced working prototypes based on the developed standards.

1.1 The Open Hypermedia Protocol

At OHS 1, it was proposed that the OHSWG could begin its work on the interoperability issue by addressing client interoperability of open hypermedia systems. An important motivation for tackling this problem first, was the observation that open hypermedia developers were spending more time on the creation of client applications to communicate with their servers than on the servers themselves, which were considered to be the important and interesting part of the research! It was argued that if the community could define a standard protocol that facilitated the communication between open hypermedia clients and open hypermedia servers, then a standard set of clients could be produced by the community that could be used with all open hypermedia systems that implemented the protocol. This protocol came to be known as the Open Hypermedia Protocol (OHP).

The first OHP proposal was introduced by Davis, Rizk and Lewis (16) at OHS 2. The proposal was heavily influenced by the protocols within Multicard (17) and Microcosm (18). The proposed protocol was conceived as being simple and lightweight. It attempted to capture only the common features of open hypermedia systems, and it was aimed at supporting hypermedia services in applications such as Emacs and Microsoft Word by taking advantage of their native scripting languages.

In the following year, simple demonstrations were built at the University of Southampton, Texas A&M University and Aarhus University. These prototypes led to the understanding that the group was more about defining *interfaces* than protocols. These interfaces might be implemented over any of a number of existing (communication) protocols.

As the community became increasingly interested in the idea of a protocol, it became clear that more was required of this effort than had originally been envisaged. For instance, it became necessary to define a common data model. This data model had to be all encompassing in the sense that it had to be possible to represent all existing hypermedia data models within this model. As the protocol developed it became clear that the original goal of producing commonly available clients that could be reused with different servers had been overtaken by the desire to achieve *interoperability* of different services. The requirement for interoperability meant that the community had to be much clearer about the architecture of the systems it was attempting to integrate.

The final evolution of the protocol occurred when it became evident that the OHSWG needed to support more than just the traditional model of navigational hypermedia. If the framework developed by the OHSWG was to be truly open, then it needed to be able to cope with spatial models of hypertext (19), taxonomic hypertext (20), hypertext by transclusion as embodied by Xanadu (21), etc. This understanding led to the design of a component-based framework consisting of many interfaces (services) at OHS 3.5 in September 1997. It was decided at that meeting to make the existing navigational protocol the first interface of the new open hypermedia framework.

1.2 Open Hypermedia Services

In the open systems literature, the term “open” is generally assumed to mean that there is some interface through which it is possible to communicate with some system or subsystem. A number of authors (7)(22) have attempted to define the term “open hypermedia”. The original OHP proposal (16) defined the term by identifying six dimensions of openness for open hypermedia systems: scalability, data formats, applications, data models, platforms, and users. The open hypermedia community has adopted the view that open hypermedia systems should provide hypermedia services to all applications on the desktop. Such systems should not be confined to simple models of hypertext, but should be extensible to work with new models, new data formats and new desktop applications. Users should not be confined to using passive browsers, but should be able to edit both content and structure (given appropriate permissions) and should be able to negotiate with open hypermedia systems about the types of links to be provided.

The World Wide Web is an open system in that it provides a protocol (HTTP) through which a browser may communicate across the Internet with a server. The use of this protocol makes it possible for third parties to produce their own browsers and servers. However, the World Wide Web has not generally been considered an open *hypermedia* system, since the hypermedia data model is very simple, and links are entirely embedded within a Web document rather than held separately (23).

On the other hand, although most of the so called "open" hypermedia systems produced by the OHS community keep links separately and provide desktop applications with hypermedia services, they have all tended to use their own (proprietary) interface to the server. Thus, in practice these systems are not "open" to client applications (browsers/editors) from other open hypermedia systems.

There is one essential decision that must be taken when implementing open hypermedia services, and that is to decide where to put the hypermedia functionality. One extreme is to put all the hypermedia services into the client; so the client will request a document, and will also request links for this document, and will place the links as "hotspots" over the document at the time that the document is displayed. At the other end of the spectrum, there are approaches which assume that there is no hypermedia functionality in the client; the client requests a document, and as the document is delivered, the server, or a proxy, inserts mark-up (typically HTML) into the document to represent the links.

The advantage of taking the server-side approach is largely pragmatic; all that needs to be installed on the client is a standard Web browser. Taking the client-side approach, it is necessary to have hypermedia-aware clients for all the data formats one intends to view. Interestingly, Hyper-G (6) and Microcosm (7) originally took the client-side approach, but then both groups came out with commercial products (HyperWave and Webcosm) which took the server-side approach to take advantage of the wide availability of Web browsers. Obviously, there are richer nuances in achieving application integration, for instance with the Web (24) or with OHSs in general ((25) and Section 4.1). However, the goal of the OHSWG is to achieve more than passive browsing of documents, but rather to provide open hypermedia services to the entire desktop. Therefore the open hypermedia framework adopts the client-side approach.

This paper is organized as follows. The following Section 2 discusses the requirements for open hypermedia services. We then look at the data model developed by the OHSWG in Section 3. Next, in Section 4, we examine the initial prototypes built using the open hypermedia framework and explain the interfaces that were developed. Section 5 addresses future work. Finally, the conclusions in Section 6 examine the benefits of having a community of researchers evolve the design of a standard protocol and reflect on the implications this work has for application developers outside of the open hypermedia community.

2 SCENARIOS AND REQUIREMENTS

From the beginning, two visions have guided the development of open hypermedia systems. The initial vision was the idea of a universe of documents (a docuverse) accessed and maintained by many heterogeneous hypermedia systems. The use of such a docuverse does not necessarily imply heavy collaboration among users. However, as Bush (26) pointed out, a natural use of hypermedia would be to share knowledge and paths through the docuverse. Thus, the vision of supporting collaborative work in and by such a shared docuverse augmented the first vision. In the following we will analyze these two visions, clarify them using some scenarios, and come up with requirements on open hypermedia systems.

2.1 Open Hypermedia Systems: The Docuverse Vision

A docuverse in the open hypermedia context can be defined as a set of information units (or resources) and a set of relations among them. Both, information units and relations, are displayed and manipulated using clients of the open hypermedia system. Usually, viewers or editors are used to display and manipulate information resources, and browsers or editors are used to display, navigate, and manipulate the relations. The open hypermedia system as a whole provides interoperability among different viewers/editors/browsers, and facilitates access to persistent information units and relations.

Let us imagine the following scenario, which has been shortened and adopted from Østerbye's "Fred the programmer" scenario (27): In this scenario, Fred is developing a software system. While using his software development environment

(SDE), he has problems in understanding one of the APIs he is using. He clicks on the name of the procedure he does not fully understand and the SDE opens a new window with the respective code (i.e. a link from the procedure name to the source code was traversed). Since he is rather interested in information about the design, he looks up the maintenance manual of the system and selects the phrase “Design Document”. By clicking on the phrase, a link engine is triggered to examine all links in the current context (i.e., the current software development and the APIs it is using). As a result, those links whose source locations include the phrase “Design Document” are displayed for selection. Among those, Fred selects the one seemingly most promising. There he gets the needed information. While continuing this process, he adds notes and links to his personal log, so he can later remember what he did to solve the problem.

In this simple scenario, the following problems become visible:

- In order to use information units and relations created by other participating systems (e.g. the SDE, the documentation browser, the link engine, and Fred’s personal log editor) a standardized format for representing structure and content is required (Requirement R1). This includes the means for expressing hypermedia constructs such as nodes, anchors and links.
- To be able to access structure and content maintained by some other OHS, there must be means for identifying and obtaining such information from this OHS (R2).

For displaying and manipulating structure and content, two requirements must be met:

- It must be possible to integrate any kind of information resource into the docuverse. As a consequence, the corresponding application or viewer must be able to be called by the open hypermedia system to display, for example, a link destination (R3).
- In addition, it must also be possible to add and update links and content within information units (R4).

In a docuverse, one might expect many different information resources structured or linked together in complex ways. Exactly how these resources are linked or structured depends heavily on the purpose and domain of the information. In the scenario above, API documentation and source code were linked to support a software development process. Fred’s personal log was used to maintain personal memory. The link engine was used to maintain and find relationships between different document spaces. From this, we can define two more requirements:

- Support for multiple structures (R5): It must be possible to support several ways of structuring or linking information resources, usually dependent on the domain and purpose of the information. This is consistent with the OHSWG’s realization that support for additional types of structure other than navigational hypermedia is required of its open hypermedia framework. Taxonomic hypermedia, spatial hypermedia, and workflow structures are considered other examples of structures, all of which should be supported.
- Support for combining multiple structures (R6): Since it is not foreseeable exactly which kind of structures are needed or need to be combined for a certain task at a certain time the need for combining (and possibly adding new) different structures arises. Thus, combining structures defined by different OHS clients (e.g., the SDE and the link engine) on the fly becomes possible.

A serious problem is the provision of a consistent user interface of the OHS functionality across different participating OHSs and applications (R7). This is an open issue first defined by Pearl (8) that has yet to be adequately addressed by the hypermedia community. It is nonetheless a requirement that can be used to influence the design of the client-side interface to the open hypermedia framework.

2.2 Open Hypermedia Systems: The Collaboration Vision

While the first vision already contains some form of asynchronous collaboration (multiple stakeholders prepare information that is used by others), it is quite clear that OHSs can and should provide support for collaboration. The following scenario is used to demonstrate some of the requirements induced by collaboration in hypermedia environments. The participants in this scenario want to collaboratively author a joint document by using their own hypermedia authoring environments. The situation can be characterized by:

- multiple co-authors, located at different sites,
- each co-author may use distinct hypermedia authoring environments,
- the co-authors need to share a common (but potentially distributed) document that is maintained in the docuverse, and
- there is a need to support asynchronous, as well as synchronous, collaboration.

Author A and author B are working on a common document that consists of a network of nodes and links that are partitioned into two subsets, one for each author. Each subset is maintained by the respective author's hypermedia authoring environment. Now imagine the following collaborative process:

1. Author A works on his local partition (i.e., browses and edits the document's components). While doing so, he might follow a link to author B’s partition, which is maintained by her hypermedia authoring environment (server). Thus, access to and display of “foreign” document parts/partitions must be possible (R8).
2. Now, author A wants to edit a part of author B’s partition (i.e., either change existing objects, or add objects). Thus, the issue of access control occurs (R9). Furthermore, the OHS of author B must now either store or at least reference objects created by A’s client application.
3. At this time, author B enters the game by opening her browser and noticing A’s presence. This means that there must be some group awareness mechanism that allows the collaborators to recognize each other’s presence (and activities). This mechanism consists of awareness detection (potentially in the server) and awareness presentation (in the client/browser) (R10)(28).
4. Author B now wants to join A’s editing session to coordinate their common efforts. Thus, B’s browser needs a mechanism to contact A’s browser and to establish a cooperative editing session (R11). In such a session, the authors want to share a joint workspace (e.g., the workspace of author A) and a joint view (potentially some relaxed WYSIWIS presentation (29)). For this, the coupled browsers or hypermedia environments need to translate their respective navigation commands into each other’s “command language”. Furthermore, the translation of different display techniques (even of a shared document) needs to be supported. Additional communication channels (such as A/V or a chatbox) also need to be provided.
5. After they conclude their joint editing, author B leaves the joint session and starts to work on her own (in a different part of the document). Thus, dynamic and spontaneous session management (creation, join and leave) must be supported.
6. Sometime later, author A finishes his work and quits his browser. Now, author B deletes some objects referenced by A’s environment. In this situation, one would expect some functionality that allows the communication of such changes between the different systems (either delayed or immediate) to ensure consistency (R12).

The above scenario is just one possible (and very short) episode within a larger collaboration process. The main requirements of the scenario include:

- Support for references between distributed document partitions (R8): Support for references from a document maintained by one hypermedia environment into another document maintained by another hypermedia environment. This includes not only the issue of addressing but also of document retrieval and display of “foreign documents” and the storage of references to remote documents.
- Support for access control (R9): To control access to documents/hypermedia objects, user authentication, and the definition of access rights, is required.
- Support for group awareness (R10): There must be some group awareness mechanism that allows the collaborators to recognize each other’s presence (and activities).
- Support for joint editing sessions (R11): Thus, browsers need a mechanism to contact other/remote browsers and to establish a cooperative editing session. In such a session, the authors want to share a joint workspace (e.g., the workspace of one author) and a joint view (potentially some relaxed WYSIWIS presentation). For this, the coupled browsers or hypermedia environments need to translate their respective navigation commands into each other’s “command language”. Furthermore, the translation of different display techniques (even of shared documents) needs to be supported. Additional communication channels (such as A/V or a chatbox) need to be provided. Dynamic and spontaneous session management must be supported.
- Support for consistency between remote document partitions (R12): If co-authors change a document partition with dependencies to other remote document partitions (e.g., removing an object that is referenced by other remote objects) then consistency should be maintained. This can be achieved by different approaches: e.g., notification mechanisms, transactions, object replication and synchronization techniques.

Table 2.1 summarizes the requirements described in the previous sections. In the remainder of the paper we will refer to these requirements via their numbers where appropriate.

Req. No.	**Description**
R1	Standardized representation format
R2	Means for identifying structure
R3	Integration of arbitrary data sources
R4	Update structural information
R5	Support for multiple structures
R6	Support for combining multiple structures
R7	Provide a consistent user interface across applications
R8	Support for references between distributed document partitions
R9	Support for access control
R10	Support for group awareness
R11	Support for joint editing sessions
R12	Support for consistency mechanisms between document partitions

Table 2.1. Summary of Requirements on the Open Hypermedia Framework.

3 UNIFIED DATA MODEL

We now present the hypermedia data model developed by the OHSWG as the foundation for the open hypermedia framework. The model is presented in an object-oriented manner (i.e., using inheritance relations). However, please note, that since any particular interface in the open hypermedia framework consists of leaf classes (i.e., classes with no children), our object orientation has no implications on the interface and no requirements for an object-oriented implementation. The data model presented in this section consists of foundational concepts such as those required for locating an object, assigning unique identifiers to objects, and connecting to a service, as well as those concepts that make up the navigational hypermedia interface (commonly referred to as OHP-Nav). As new interfaces are added to the framework, this data model may be specialized to handle constructs such as composites, spatial hypermedia structures, and work flow specifications. Before we discuss the data model specifically, we first present the OHSWG's assumptions about the architectural context of the open hypermedia framework.

3.1 Data Models and Interoperability

As stated above, support for interoperability has been the underlying motivation behind the work of the OHSWG (30)(31). In (30), a common reference architecture is proposed for the open hypermedia framework that identifies two main interfaces between three conceptual layers (see Figure 3.1). The layers consist of the content handler layer (referred to above as client applications), the open hypermedia framework layer (consisting of a set of interfaces), and a hypermedia database layer (also called the backend). The two interfaces are the content handler interface (CHI), which is the realization of the initial open hypermedia protocol (OHP), and the hypermedia database interface (HDBI), which is a proposed interface for enabling communication between hypermedia services and hypermedia databases. This latter interface has not yet been the focus of the OHSWG's efforts, but the idea is that it should support the general storage of hypermedia objects and provide support for collaboration and distribution in a standardized way.

A standardization of the CHI enables interoperability between tailored content handlers and hypermedia services. This implies that a content handler can be tailored and then it should work with any open hypermedia service supporting the standard CHI. For the rest of the paper, we will use the term OHP (open hypermedia protocol) instead of the term CHI (content handler interface) for purposes of uniformity. Note that the open hypermedia protocol depicts one possible physical realization of the standard CHI (see also Section 5.3.2).

A standardization of the HDBI enables interoperability between hypermedia services and hypermedia databases. This implies that collaboration support, for example, can be developed once and provided for many different types of hypermedia services compliant with the HDBI. In (32) a number of different types of (non-traditional) hypermedia services is proposed.

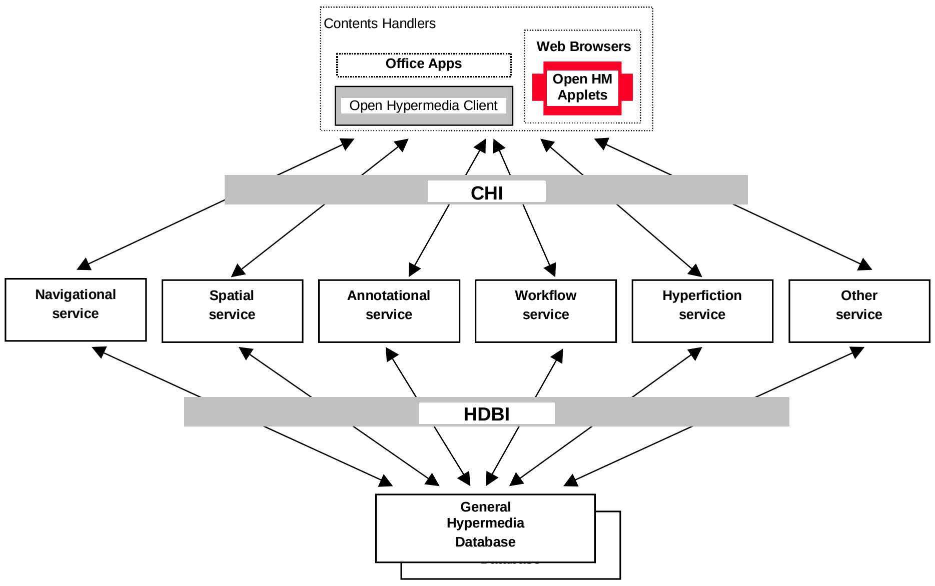


Figure 3.1. Interoperability among hypermedia functional services via common HDBI (32).

The OHSWG initially placed a significant amount of effort into the design of the OHP as a protocol. However, exclusive focus on the protocol aspects would ultimately lead to failure. Imagine if the WWW had only standardized HTTP and left HTML unspecified. In that case, the Web would have a standard means for communication between clients and server but no means for understanding the information being communicated! The OHSWG found itself in exactly that situation. The protocol had standardized operations such as *NewLink*, but it had not addressed the fact that each open hypermedia system may have its own semantics for that operation. Without standardizing semantics, a user creating a set of structures using one OHS would discover that he could not combine these structures with the structures of another OHS, even though both systems may have implemented the OHP. This scenario is not far-fetched. Consider the case where a user comfortable with the guided tour support of one OHS discovers that his tours cannot include content managed by another OHS. As a result, it became clear that the OHP needed to be supplemented with an explicit understanding of the underlying data model in compliance with requirement R1.

At OHS 3.5, the HDBI was introduced as one approach to achieving this explicit understanding. The HDBI allows multiple hypermedia services to access and share different types of hypermedia structures in a standardized fashion (as depicted in Figure 3.1). We now present the data model of the OHSWG.

3.2 An Extensible Open Hypermedia Data Model

The independently developed data models of various OHSs have a lot in common. For instance, an analysis of the Dexter model extensions (33) and data models implicitly or explicitly formulated in OHP (16) as well as various open hypermedia systems (2)(3)(4)(5)(34) reveals many similar concepts. The OHSWG's unified data model represents a synthesis of these former approaches, specified in an object-oriented manner with agreed upon class names.

Interoperability requires careful consideration of the data models to be shared among services. There are two extreme solutions spanning the range of design possibilities:

- Unified non-typed objects in which all properties are specified using dynamic attribute-value lists;
- Strongly-typed objects with pre-defined semantics.

With solution 1, every service can specify its own encoding of specific data models into attributes and values. This is a very open model that can handle most services that will be devised in the future. However, an important drawback is that the encoded model may not be understood by any of the other existing services. Moreover, the specific data model of the hypermedia service has to be encoded and decoded by the programmer who writes the service.

With solution 2, all hypermedia services will know exactly what the data model is and what the semantics associated with it are. However, a drawback here is that new types of services may impact the data model such that changes are required. These changes then have the potential to affect all previously existing services and stored structures.

The OHSWG determined that an approach in between 1 and 2 is needed to provide interoperability among open hypermedia services while ensuring openness with respect to new types of services. Therefore, the key issue is to design a common but extensible data model. This provides the basis for supporting multiple structures and their combination (requirement R5). The OHSWG settled on a framework, in which the primary characteristics of objects are handled as first-class attributes and the secondary set of characteristics are kept open for future extensions by means of attribute values.

3.3 Core Data Model of the Open Hypermedia Framework

The basic element of the core model is the abstract class *AbstractObject*. An abstract object consists of an identifier, a name, a set of descriptions, a type, and a set of characteristics. A characteristic consists of a name (String) and a set of values (Strings).

The basic classes and their specialization into a navigational data model are depicted in Figure 3.2.

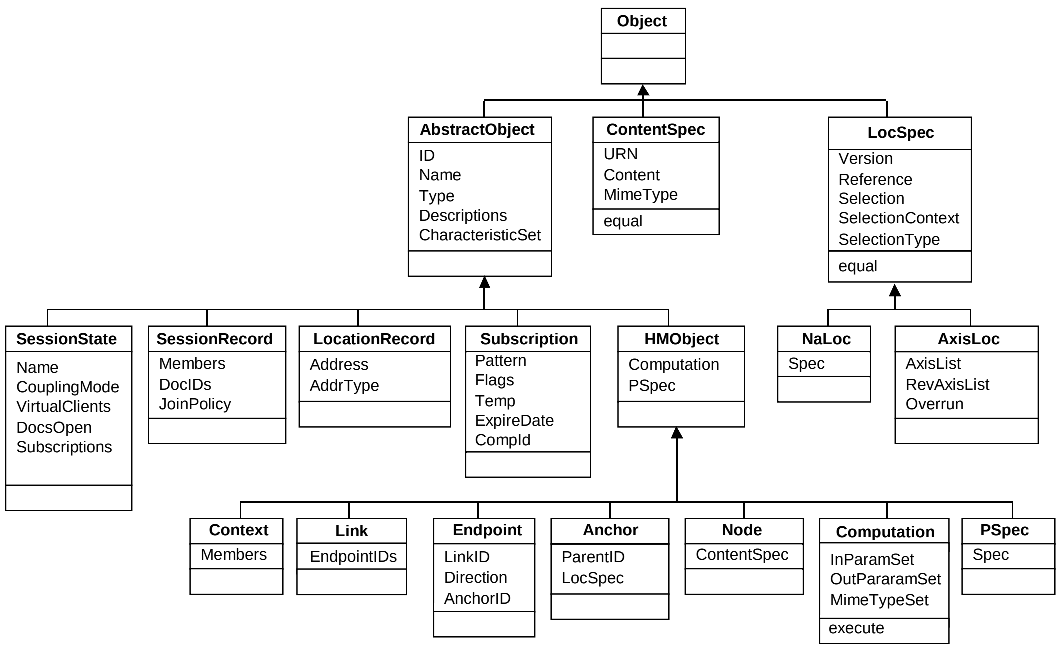


Figure 3.2. OHSWG core data model — generalization.

The specialization hierarchy in Figure 3.2 has a general subclass for all hypermedia objects that need a globally unique identifier, a name, a type and a set of characteristics. This class is called *HMObject* and it is considered an abstract superclass that generates no instances. Immediately inheriting from the HMObject class are all the first-class hypermedia objects that contain unique identifiers. These objects are typically associated to each other through one-to-one, one-to-many, and many-to-many relationships as depicted in Figure 3.3. The HMObject subclasses cover the classical object types such as *Node*, *Link*, *Anchor*, *Endpoint*, and *Context*. HMObject is a unification of a number of different hypermedia objects corresponding to the Dexter model “Component”, which was a unification of nodes, links and composites. HMObject, however, covers a wider range of objects including, for example, anchors and endpoints. This implies that more entities in the core data model are assigned unique identifiers in the same name spaces and thus can be addressed as first-class objects. Thus, we get at least the generality of the Dexter model, in which links can have endpoints pointing into both nodes and links. We also get a wider range of possibilities (perhaps not all equally meaningful), e.g., linking into Contexts becomes possible.

Anchors contain a *ParentID*, which refers to an object of type HMObject ID; this allows the generality described above, in which both a link and a context may be the parent of an anchor where the LocSpec may locate, e.g., an endpoint in a link or a member in a context.

Context is a class of objects similar to Hypertexts in Dexter (35) and Contexts in (36). These HMObject subclasses also include two classes used for specifying behavior at runtime: *PSpec* and *Computation*. PSpec is a specification that typically stores an opaque specification of attributes specific to a particular content handler that govern the presentation of a hypermedia object at runtime. A computation is a class of objects that store code in some programming language to be executed at runtime by some interpreter or virtual machine.

The specialization hierarchy in Figure 3.2 includes a number of specification classes: *ContentSpec* and *LocSpec*, and various LocSpec subclasses, which are used to instantiate specification objects that typically become in-lined in node or anchor objects. Thus, these objects contain no global unique identifier. ContentSpec is used either to address or to contain the actual content of a hypermedia node. LocSpec is a class modeling the location specifier concept introduced in (33). A LocSpec is used to specify a location within content specified by a ContentSpec; a LocSpec thus corresponds to the anchor value in the Dexter model (see also Section 4.2.1 below).

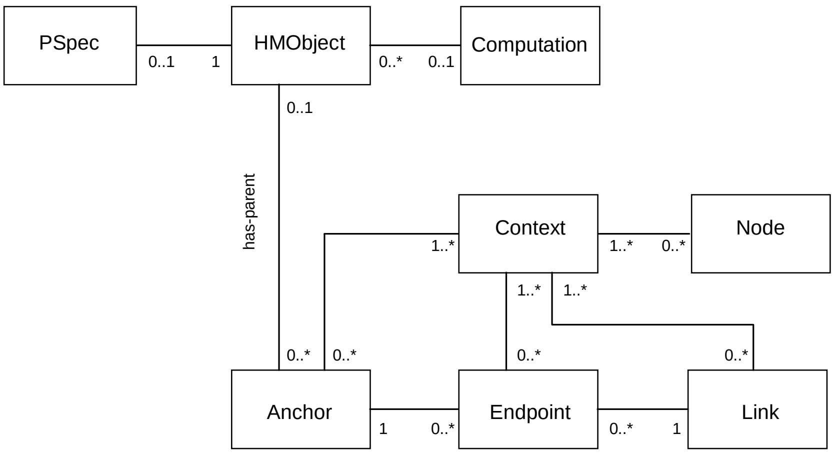


Figure 3.3. OHSWG core data model — association.

The core data model of the open hypermedia framework is open in several respects.

- The specified classes of objects may be specialized into new classes. Objects instantiated from these new classes may be associated with existing objects or new objects.
- Attributes can be added to existing types through the general characteristics set in HMObject, which is a set of attribute-values.
- Computations and PSpecs can be associated to arbitrary hypermedia objects, thus possibly extending their behavior in multiple ways.
- Finally, the model provides a degree of typing that allows a shared and common semantics across hypermedia services.

Thus, the core data model addresses the requirements R1 and R2 via the LocSpec concept, and requirements R5 and R6 since multiple structures can be modeled and combined using subclasses of the data model. Furthermore, the integration of arbitrary data sources is supported by the concept of nodes and LocSpecs (R3).

Having presented the core data model and its architectural context, we will now describe the types of systems that can be constructed using the open hypermedia framework.

4 ARCHITECTURE AND SERVICES

In this section, we examine, in depth, the architecture of the open hypermedia framework. Recall that the OHSWG settled on an approach whereby hypermedia services are made available via a component-based framework. This has led to the design of component-based open hypermedia systems (CB-OHSs). We describe the conceptual foundations of these systems next, before presenting the rationale behind a number of open hypermedia services. We then conclude this section by describing two concrete implementations of the conceptual architecture and subsets of the OHSWG defined services.

4.1 Conceptual Architecture

The environment of a CB-OHS is conceptualized as a standard three-layer architecture (see Figure 4.1). The backend consists of infrastructure for both frontend and middleware entities. The middleware layer consists of a set of conceptual servers, each of which provides an open service (to applications and/or other middleware servers). The frontend consists of applications (described as content handlers above) that may either be natively compliant (i.e., originally built to operate in the environment) or third-party clients that have been encapsulated in a compliant wrapper. We discuss the infrastructure, the open services, and the integration of applications in more detail below.

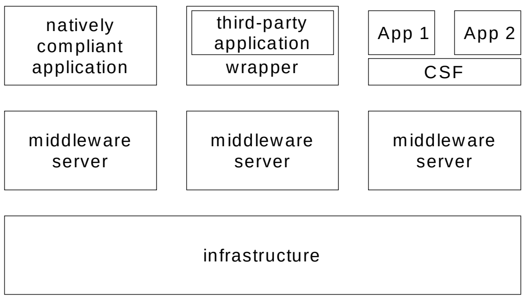


Figure 4.1. Conceptual architecture of a component-based open hypermedia system.

4.1.1 Backend

The backend is left underspecified by the OHSWG, but is generally regarded as providing at least persistent storage functionality. Middleware servers (and frontend applications) may use this persistent storage facility for both traditional data abstractions (e.g., document, file, etc.) and structural abstractions (e.g., link, anchor, etc.). Optionally, other generic infrastructure support is often regarded as residing in the backend, such as naming and location services, but these services are left undefined or partially defined, since they do not need to be fully specified for middleware services to be defined.

4.1.2 Middleware

The middleware layer consists of an open set of servers, each of which implements some number of services. Middleware servers mediate interaction among backend, frontend, and other middleware servers. Processes requiring a service must first locate a server (to this point, in an idiosyncratic manner) that implements this service. Such a server may in fact implement several other services as well, but this is not normally relevant to clients. Therefore, clients do not directly engage servers for service discovery — this is mediated by some (underspecified) location service thought to reside conceptually in the backend. Details about specific services defined by the OHSWG are provided in Section 4.2.

4.1.3 Frontend

Much research has revolved around specification of the frontend of open hypermedia systems. Some have argued that the composition of frontend entities, like the backend ones, should not be a specific concern of the OHSWG (31), while others have elevated certain individual parts of the frontend to first-class status, such as the "client-side function" (CSF) (37).

Here, we use the term *frontend entity* to describe one or more client-side processes that cannot be distributed from one another. This should be taken to include the application itself plus some optional additions, such as wrapper processes, process invokers, etc. (38). A wrapper mediates interaction between a middleware sever and an application (39). A process invoker is a special frontend entity that invokes applications and/or wrappers upon request of a middleware server (e.g., to show the result of a link traversal). Other types of frontend entities include user-interface enhancements designed to provide consistent access to services across multiple applications (e.g., the LS Menu of HOSS (4)), and a communications multiplexer that provides a common link between all applications on a given machine and all the servers they require (37).

These entities are shown in Figure 4.2. The Figure can be interpreted as a more detailed description of the frontend layer of Figure 4.1.

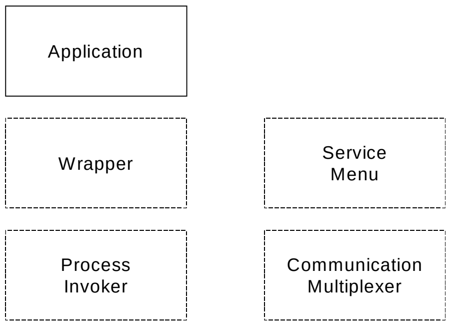


Figure 4.2. All implementations have applications that comprise (some of) their frontend entities. Optionally, these applications may be wrapped. Some implementations may provide process invokers, dedicated service menus, and/or communication multiplexers.

Allowing an open set of applications implies both the ability to custom-build an arbitrary number of such applications and to integrate existing third-party applications (18)(25) by extending or modifying them to use the open service. There are two basic ways in which applications are normally modified to use an open service. One of these is *direct extension*. In the direct extension approach, the application's native extensibility mechanism (e.g., Visual Basic in Microsoft Word) is used to program a new application extension that can interact with the service. Clearly, only applications with such a native extensibility mechanism are candidates for direct extension.

Another standard method for enabling access to open services is by introducing a *wrapper program* (or simply, *wrapper*). A wrapper is a second process that interacts with the application process through its normal input channels. That is, it essentially simulates input in order to interact with the application. It can be thought of as an application itself that is natively compliant with the open service. Some systems have developed wrappers that interact with more than one application (e.g., (2)(3)(34)).

In practice, a combination of these approaches is often used, by building a wrapper that interacts with an application extension. In such cases, the wrapper often functions to simplify and specialize the generic service interface. Then, a hopefully simpler application extension is written to interact with this specialized interface.

The mechanisms proposed in this section support the integration of arbitrary data sources and viewers as requested by requirement R3.

4.2 Conceptual Service Definition

Members of the OHSWG have been working on the definition of different service interfaces such as navigational hypermedia services, spatial hypermedia services, taxonomic hypermedia services, composition services, collaboration services, and content-based retrieval services. This section describes three of the service interfaces that have reached a level of maturity that allows development of (de-facto) standardized middleware services.

4.2.1 Navigational Hypermedia Services

By hypermedia services, we mean so-called “navigational” or “associative” hypermedia, as described originally by Bush (26). Navigational hypermedia has since then been the most widespread and common type of hypermedia. (For example, the WWW implements a kind of navigational hypermedia.) The ideas behind navigational hypermedia grew out of observations by early pioneers in the field such as Bush (26), Engelbart (40), Nelson (41), and others. They noted that people use a type of idiosyncratic associative storage and recall scheme within their own memories. That is, people form idiosyncratic “links” among different items. Recall of a given item can result in easier recall of the “linked” (associated) items.

Navigational hypermedia systems have been built to support such associative storage and recall functionality with data that are stored in computers rather than inside people's memories. There are many examples of navigational hypermedia systems, and of open hypermedia systems that provide support for navigational services. The OHSWG’s navigational interface, OHP-Nav, has been under development since early 1996 (16)(42). The first visible result of this effort was a software demonstration at the 1998 ACM Hypertext Conference in which several open hypermedia systems showed true interoperability at the application level using the developed OHP-Nav standard.

The most important abstractions of the OHP-Nav service are described below. (The following description is meant to supplement the information presented in Section 3.3).

A *Node* provides a “wrapper” for an arbitrary resource (addressing R3). It provides the hypermedia service a “proxy object” for this resource by means of a content specifier (*ContentSpec*). One example of a ContentSpec might be a Uniform Resource Name (URN), although the hypermedia service does not interpret the contents of a ContentSpec, and, therefore, it may have arbitrary content.

An *Anchor* consists of an identifier and an (optional) location within the identified object. This location is provided by means of a location specifier (*LocSpec*). A LocSpec provides a handle to some part of a resource. It is used to specify a location in arbitrary content in a client-specific fashion. Examples of a LocSpec may be, for instance, a byte offset and byte extent in a stream of binary data, or a pair of (x, y) coordinates in an image file. The identified object may be a node identifier, or the identifier of any other open service abstraction.

An *Endpoint* consists of an anchor identifier and a link identifier. It binds exactly one anchor with exactly one link, but both anchors and links may appear in multiple endpoints. A link, as just stated, is referred to by some number of endpoints. A context consists of an arbitrary set of links, endpoints, anchors, and nodes.

These abstractions might be used in the following way. Suppose that a client wishes to associate some passage in a text file with some part of an image. It could create two nodes — one to wrap the ContentSpec for the text file and one to wrap the ContentSpec for the image file. Then, it could create two anchors — one to point to the text file node (with a LocSpec designating the appropriate passage) and one to point to the image file node (with a LocSpec designating the area within the image file). Finally, it could create an endpoint for each anchor and a link that is referred to by each of these endpoints. These objects could be placed within any context. The hypermedia service can then provide a *FollowLink* service to clients, returning the ContentSpec and LocSpec of the area with the image file when the follow link service is invoked on the ContentSpec and LocSpec of the passage in the text file (or vice versa). This way, arbitrary application data can be linked (R3).

4.2.2 Collaboration Services

We assume the clients of the collaboration service to be entities that themselves mediate interaction among multiple clients or users. That is, we normally see a collaboration service as something provided to “collaboration-aware” clients, and only in special cases as something that allows multi-user collaborative work among otherwise single-user or non-collaborative entities. We envision a collaboration service that provides two basic abstractions: *Session Record* and *Session State* (see Figure 3.2).

A session record contains basic information about a collaborative work session, such as its name, its members (and their current status — *logged in* or *logged out*), the tools currently used by the members, the documents currently shared, the coupling mode, the joining policy, and one or more communication addresses with which new potential members can communicate. Session records are created by clients of the collaboration service. Other entities can then query the collaboration service for session records that match given criteria, obtaining the appropriate communication address(es). However, session records do contain an open set of multi-valued attributes that can be used to store client specific information.

A session state contains other information about the session such as current pointer locations and “catch up” information for clients that join the session late. Each member of a session has a distinct copy of the session state. Additionally, the entity managing the session (i.e., the client of the collaboration service) also maintains a distinct copy of the session state. Inspired by the GMD-IPSI work in this area (such as CHIPS (43)(44)), fields in a session state may be in one of two modes. If a field is *shared*, all reads or writes of that field made on a member copy of the session state will be redirected to the managing entity. If a field is *not shared*, all reads or writes on a member copy take place on that copy itself. Additionally, writes to a member copy may optionally have an effect on the managing entity. Non-shared fields may be resynchronized by copying the managing entity’s field value to all member copies.

Either the session record or the session state may be made partially or fully persistent. (Making an abstraction “partially persistent” means that only some of its fields are made persistent.) Both of these abstractions are bridge abstractions as described above, since the collaboration service does not enforce any semantics on the field values of the abstractions. That is, clients may place any information they deem useful in the fields. Because this information is left uninterpreted, these abstractions do not replace any existing client abstractions, thus allowing this service to be provided orthogonally.

Using session record and session state, clients can manage joint editing sessions (R11). Via shared fields clients may implement group awareness (R10). Consistency between document parts (R12) and access control (R9) could be implemented in the backend. The core data model supports distributed documents (R8).

4.2.3 Content-based Retrieval Services

Hypermedia systems often allow their proprietary clients access to advanced functionality, in particular for supporting navigation within multimedia documents (45). To enable this functionality to be accessed by generic services it is necessary to support the abstract notion of a computational service. A computation is a black box of functionality, known only by name to a client, that can be invoked and its results understood, even though its workings are completely opaque. In this way a generalized client gains access to complex functionality that would otherwise be unavailable.

The need for such computations arises in situations where clients have to be lightweight but still want to support advanced functionality to the user. For instance, a client application may support navigation by finding “similar” images, where the notion of similarity may depend on different characteristics such as the color space, color distribution, or other meta-data of the image. Clients can offer this functionality to the user by relying on external services which are dynamically discovered and invoked and which they only need to call rather than implement themselves. Thus, lightweight clients are able to offer advanced functionality such as content-based navigation.

A *Computation* is defined by its name, a set of input parameter templates, a set of output parameter templates, and a set of mime types for which the computation is valid (see Figure 3.2). Any component interested in using computations can retrieve the set of available computations by using functions for retrieving such a list from other components in the system. It can itself make computations available to the system as well. Computations with parameters which are not understood can simply be ignored.

Often computations may actually take a long time to complete, resulting in the obvious problem of giving feedback to the user. The definition of the retrieval interface therefore includes a mechanism which lets the calling component know the time that a service takes and which allows the called component to send progress messages back to the caller to give explicit feedback.

So far we have described what we believe to be a powerful mechanism for components to access hypermedia services but it is conceivable that this mechanism will not be powerful enough. For instance, for content-based navigation in audio (46), a computation might analyze a selection of music and produce a contour (a simplified representation of the selection used for comparison purposes); it would then check that contour against other contours in a database to produce a list of nodes representing musical scores similar to the selection. The issue here is that although the input and output parameters of this complex computation are well defined the computation itself needs to be split up into several sub-computations for scalability and performance reasons. Clearly, the analysis of a huge audio file should take place at the file’s location (i.e. usually at the content handler). Matching the resulting contour against a database however, could be performed elsewhere. We have therefore come up with the definition of a composite computation.

Composite computations contain similar information to computations but are not called directly, instead they include a computation graph. This describes how computations can be combined, allowing them to be called in serial or in parallel. By having a non-cyclic graph we avoid problems with managing the flow of data (such as getting stuck in loops) while retaining significant computational power.

Composite computations represent a way of a component “imparting knowledge” to the rest of the system about how computations can be combined. This leaves plenty of scope for the development of more sophisticated systems which learn about particular combinations of computations and also provides a framework for the exploration of mobile code and agents within a CB-OHS environment.

4.3 Implementations

Based on the conceptual architecture described above, a couple of prototype implementations have been developed. In the following we will describe the Construct system and the Solent system.

4.3.1 Construct

Construct is a CB-OHS under development at Aarhus and Aalborg Universities in Denmark (47). This work builds on previous open hypermedia services research conducted over the last decade (e.g., HOSS (4), HyperDisco (5), and DHM (48)). Construct is shown in Figure 4.3. Analogous to the conceptual architecture above, the Construct architecture consists of three layers. Each is described in more detail below.

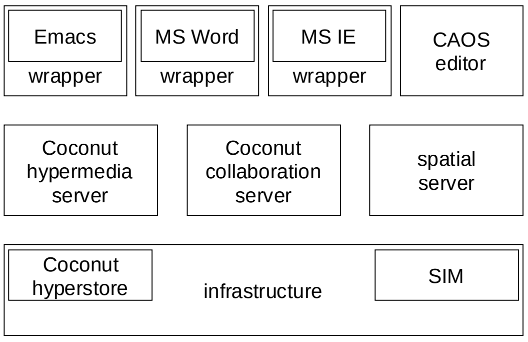


Figure 4.3. Construct prototypic implementation of the OHSWG conceptual architecture.

4.3.1.1 Infrastructure

The Construct infrastructure layer encompasses two types of components, the hypermedia store (hyperstore) and the service information manager (SIM). The hyperstore is a structure-aware backend that handles persistent storage and provides generic backend services to an open set of middleware services. The hyperstore implements core hypermedia services (such as atoms, attributes, behaviors and relations), and core collaboration services (such as transactions, concurrency control, notification control, access control, and version control). The SIM provides naming and location services that allow Construct components to operate in a massively distributed setting such as the Internet.

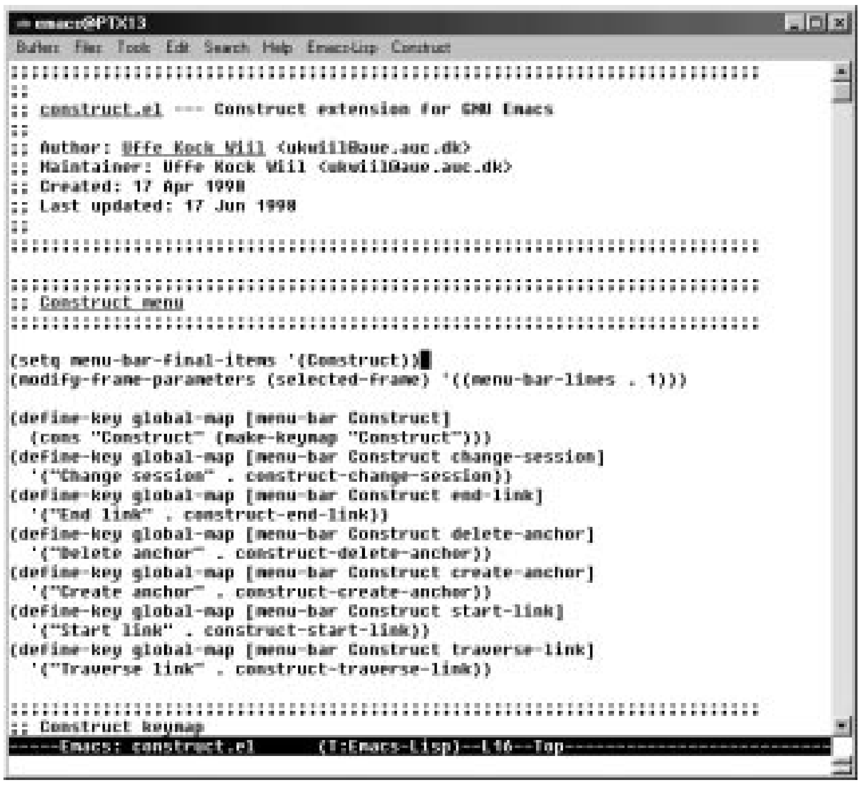
4.3.1.2 Middleware services

Currently, the middleware layer contains three servers. Two of these were developed as part of the Coconut project (<http://www.cit.dk/coconut/>). The Coconut hypermedia server implements the navigational hypermedia service described above, plus a composition service. (The composition service is described in more detail in Section 5). The Coconut collaboration server implements the collaboration service described above. A server implementing the content-based retrieval service is partially complete — work is continuing on this front. The spatial server implements a hypermedia service (briefly described in Sections 4.4, 5.2 and 6.2).

4.3.1.3 Applications

A number of applications have been integrated or custom-built to interact with Construct middleware components. Some of these are shown in Figure 4.3; others are mentioned or discussed below.

Emacs is a widely used text editor available on a number of platforms including Unix and Microsoft Windows. It is a powerful editor with its own built-in extension language (Emacs Lisp). Emacs was integrated with Construct using a wrapper written in Java. The wrapper communicates with Emacs using a simple text based protocol on top of TCP/IP. The wrapper translates the requests from Emacs into OHP-Nav operations and sends them to the Coconut hypermedia server. Responses from the server are translated back to the text-based protocol and sent to Emacs. Emacs itself has been extended with three kinds of Emacs Lisp functions: those that provide the user interface to the navigational hypermedia services, those that handle responses from the wrapper and those that handle communication to the wrapper. An Emacs Lisp file with the extension functions is loaded at start-up and the navigational hypermedia library is installed in Emacs. Figure 4.4 shows a screen shot of Emacs when running as a Coconut hypermedia server application.

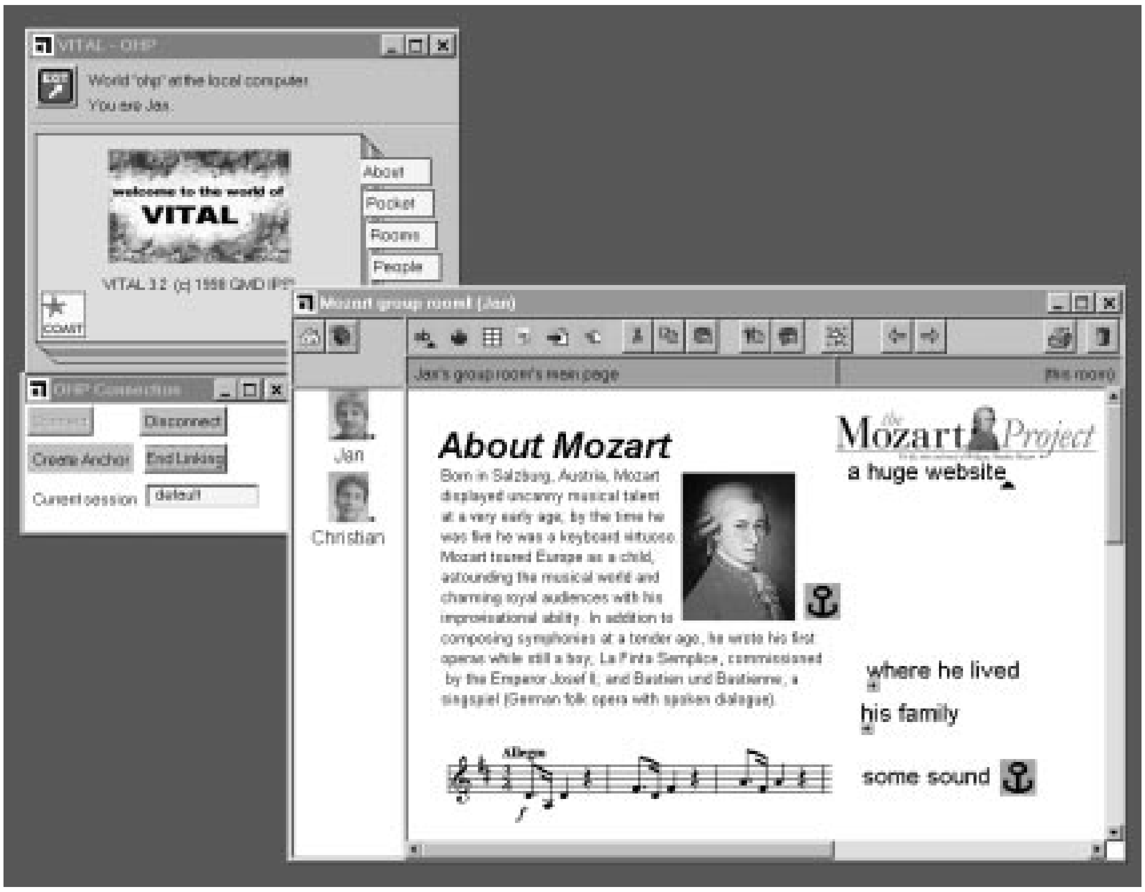


```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; construct.el --- Construct extension for GNU Emacs
;;
;; Author: Uffe Kock Will <ukwill@aue.auc.dk>
;; Maintainer: Uffe Kock Will <ukwill@aue.auc.dk>
;; Created: 17 Apr 1998
;; Last updated: 17 Jun 1998
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Construct menu
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(setq menu-bar-final-items '(Construct))
(modify-frame-parameters (selected-frame) '((menu-bar-lines . 1)))

(define-key global-map [menu-bar Construct]
  (cons "Construct" (make-keymap "Construct")))
(define-key global-map [menu-bar Construct change-session]
  '("Change session" . construct-change-session))
(define-key global-map [menu-bar Construct end-link]
  '("End link" . construct-end-link))
(define-key global-map [menu-bar Construct delete-anchor]
  '("Delete anchor" . construct-delete-anchor))
(define-key global-map [menu-bar Construct create-anchor]
  '("Create anchor" . construct-create-anchor))
(define-key global-map [menu-bar Construct start-link]
  '("Start link" . construct-start-link))
(define-key global-map [menu-bar Construct traverse-link]
  '("Traverse link" . construct-traverse-link))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Construct keymap
-----Emacs: construct.el      (T:Emacs-Lisp)--L16--Top--------------------------
```

Figure 4.4. Emacs as a Construct application. OHP-Nav anchors are shown as underlined text.

A new menu (named “Construct”) with five navigational hypermedia menu items (Traverse Link, Start Link, Create Anchor, Delete Anchor and End Link) is added to the Emacs interface. This is the only visible indication that Emacs has been extended with navigational hypermedia services. These five interface functions are also available via the keyboard.

Vital (49)(50) is a hypermedia-based collaborative learning tool developed at GMD-IPSI using the Coast toolkit (51). Vital is developed in Visual Works Smalltalk, which runs on both Sun Sparc and Microsoft Windows platforms. Vital is integrated in much the same way as Emacs. A wrapper written in Java performs transformation operations similar to the Emacs wrapper. A Smalltalk code library extends Vital with a menu and the necessary communication functions. Figure 4.5 shows a screenshot of Vital when running as a Construct application.



Born in Salzburg, Austria, Mozart displayed uncanny musical talent at a very early age, by the time he was five he was a keyboard virtuoso. Mozart toured Europe as a child, astounding the musical world and charming royal audiences with his improvisational ability. In addition to composing symphonies at a tender age, he wrote his first operas while still a boy, La Finta Semplice, commissioned by the Emperor Josef II, and Bastien und Bastienne, a singspiel (German folk opera with spoken dialogue).

Figure 4.5. Vital as a Construct application. Vital anchors are shown as arrow icons and OHP-Nav anchors are shown as anchor icons attached to the linked pieces of information.

A new window with the available navigational hypermedia functions (Connect, Disconnect, Create Anchor and End Linking) is added to Vital's interface (see small window on the left side of Figure 4.5). Vital links span pieces of information internally in Vital, while OHP-Nav links connect Vital items to pieces of information outside of Vital (e.g., to some words in a text file displayed by Emacs).

4.3.2 Solent

The Solent system is a CB-OHS which takes its name from the waters around Southampton. Solent builds on the experience gained in developing the many previous open hypermedia systems and prototypes at the Multimedia Research Group at the University of Southampton, including Microcosm (7), Microcosm-TNG (52), Mavis (45), and MEMOIR (53). Following the architecture shown in Figure 4.1, the conceptual architecture of the Solent system is shown in Figure 4.6 below. We will describe each of the layers in more detail in the following subsections.

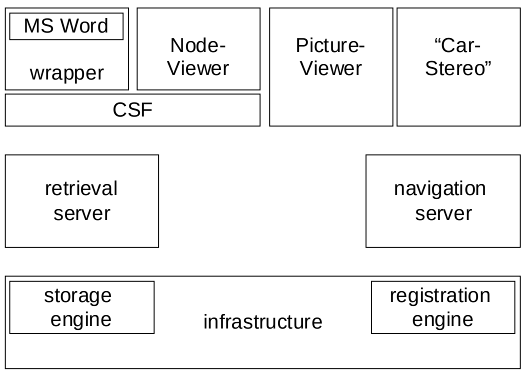


Figure 4.6. Solent prototype implementation of the OHSWG’s conceptual architecture.

4.3.2.1 Infrastructure

The Solent system is composed of communicating components, each of which may contain one or more engines. An engine could best be described as a software process managing a certain functionality within the system. The core of the infrastructure consists of a basic broker object which by default only contains a registration engine with which other components can register. Hence, strictly speaking, the storage engine actually belongs to the middleware server layer. The storage engine provides storage and retrieval of arbitrary structures that are encoded in XML (eXtensible Markup Language). It can be configured to communicate with any database supporting SQL.

4.3.2.2 Middleware Services

Solent is the latest in a series of prototype OHP servers developed in Southampton. Initially the system was based upon a version of Microcosm which had been “shimmed” to be OHP compliant; this was used to prove the concept of the original draft proposal (16). With respect to the architectural diagram above, this could be displayed as a combined storage/navigation server.

This was replaced by a more modular implementation. Initially the Solent architecture assumed the presence of some extra client-side functionality (CSF) that acted as a router and knowledge store to the client side components (see also Section 4.1.3). This assumption was later dropped resulting in the version that exists today. Solent is based on a middleware layer that contains two main servers, the navigational server and the retrieval server; the interfaces to these two servers are described in Sections 4.2.1 and 4.2.3, respectively.

4.3.2.3 Applications

Two separate sets of applications have been written for the Solent system. The first used a CSF as a nexus for all client side communication and was comprised of a “Picture-Viewer”, a “Node-Viewer” and a “Link-Editor”; a copy of Microsoft Word was also integrated using Microsoft’s COM technology. The Picture-Viewer was a stand-alone content handler (displaying JPEG and GIF images) while the remaining applications were integrated with the CSF to enable easy browsing and modification of the information space (see also top layer in Figure 4.6).

More recently applications have been written to exploit not only navigation but also the computational aspects of the system by implementing the content-based retrieval service interface (see Section 4.2.3). A “Car-Stereo” application has been written that queries the server for particular computations and enables its interface if and when they are available. It acts as a browser for music files (MPEG-3) within the system and allows the user access to computations such as *FindSimilarSongs*. In addition, the Microsoft Media-Player was integrated to support navigation and dynamic computation discovery and invocation. This application queries the computational server for all the available computations and allows the user to access those whose parameters and mime types it understands. The user interface is realized by adding a “Services” menu which allows users to select and invoke the various computations. These applications are presented in Figure 4.7.

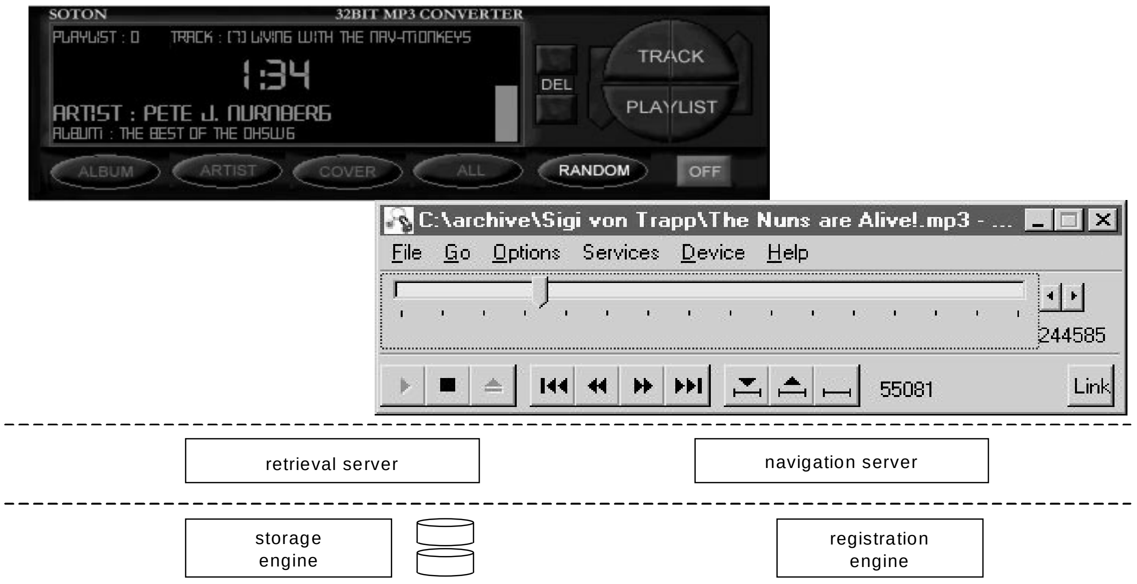

SOTON
32BIT MP3 CONVERTER
PLAYLIST : 0
TRACK : [1] LIVING WITH THE NAV-MONKEYS
1:34
ARTIST : PETE J. NURNBERG
ALBUM : THE BEST OF THE OHSWG
DEL
TRACK
PLAYLIST
ALBUM
ARTIST
COVER
ALL
RANDOM
OFF
C:\archive\Sigi von Trapp\The Nuns are Alive!.mp3 - ...
File Go Options Services Device Help
244585
55081
Link
retrieval server
navigation server
storage engine
registration engine

Figure 4.7. Solent prototype implementation of the OHSWG's conceptual architecture. The Figure shows the "Car-Stereo" application (top) and an adapted version of the Windows Media Player (below) accessing various hypermedia services.

4.4 Demonstrations

Both systems, Construct and Solent, were demonstrated at the 1998 ACM Conference on Hypertext and also more recently at the 1999 ACM Conference on Hypertext. The aim of the Hypertext'98 demonstration was to show client interoperability between different OHSs for the first time.

Initially, the Solent Picture-Viewer client was shown working with the Solent navigational server; it was then re-run this time connecting to the Construct server. In addition the Construct Emacs client and also an integrated version of Microsoft Internet Explorer were run and connected to the Construct server; all three viewers loaded the same node. An endpoint was created in the Solent Picture-Viewer and then a second endpoint in the Emacs client. The Solent Link-Editor was then used to author a link between endpoints and the link was followed using Internet Explorer.

This demonstrated not only the ability of three individually written clients to access the same server through a standard interface (addressing requirements R2-R4) but also demonstrated some basic collaboration techniques that were later expanded and refined at Hypertext'99 (addressing R11).

Since the group had already shown interoperability, the demos at Hypertext'99 were taken as an opportunity to demonstrate the new features of the interfaces and also show the work that had been undertaken in spatial hypertext. Hence, there were three demonstrations.

The first, using the Construct servers and clients, showed how the very basic session support demonstrated at Hypertext '98 had been expanded into a more full-featured interface (see also Section 4.2.2). The second, using the Solent link server and audio clients, demonstrated how specific and generic clients might use the computational interface (see Section 4.2.3) to gain access to advanced functionality. The third, which was based on the CAOS system (Collaborative Authoring using Open Spatial Hypermedia)(54) demonstrated how the same approach that had been applied to produce the navigational interface might also be applied with the aim of providing spatial hypertext functionality within the open hypermedia framework.

All demonstrations were extremely successful, with many people interested in the interoperability work of the OHSWG. They also served a useful secondary function, giving us direct experience of how to go about writing OHP-compliant system components. Basically, the following steps are required:

1. Identify the interfaces that you wish to support, e.g. navigation, collaboration, retrieval;

2. Write suitable handlers (either within the component process or as external wrappers);

3. Select and implement a subset of operations you wish to support that gives you the functionality required, e.g. for session support or invocation of computations.

Experience on adapting new content-handlers has suggested a very short time frame of only a few days to adapt an existing client to work with the interfaces of the open hypermedia framework. For instance, the integration experiment with the GMD Vital system was completed rapidly over the course of two days (see Section 4.3.1.3 above).

5 FUTURE WORK

The evolving view of open hypermedia systems as a result of the OHSWG's work addressing interoperability, has led to the realization that open hypermedia services are not only useful to client applications but to the various components of OHSs themselves. This improved understanding has resulted in a number of standardized interfaces that were discussed in Section 4.2. A natural question to ask in response to this development is what other services would it be beneficial to standardize in a similar fashion. Additional issues involve modularizing the components of OHSs further, identifying additional interfaces in the conceptual architecture to standardize and iterating on the low-level design of the open hypermedia framework. We discuss the OHSWG's stance on these matters below.

5.1 New Interfaces

There are different hypermedia domains that require inclusion at the service level of the conceptual architecture. The approach taken by the OHSWG in order to identify these domains is based on the description and analysis of scenarios (see Section 2). Also, the identification of these domains and their prototype implementations have lent insight into the nature of non-middleware interfaces — that is, interfaces at different architectural levels that could, for instance, service multiple domains.

With respect to the former, the list of potential application domains is unlimited. However, within the field of hypermedia research, there are a number of specific problem domains that have emerged over time. These include, for instance, argumentation support (e.g., (55)), taxonomic work (e.g., (20)), workflow support (e.g. (56)), and hypermedia art (e.g. (57)). Furthermore, the rather broad domain of navigational hypermedia could be extended to support specific application areas such as guided tours and trails (26)(58)(59), graphical visualizations of a hypermedia context, and composite hypermedia objects.

5.2 Non-middleware Interfaces

The set of interfaces developed and standardized so far addresses presentation interoperability, i.e., it focuses on interoperability between content handlers and domain specific middleware components (47). The main benefits of this approach lie in allowing different clients to interoperate with different middleware servers.

However, there are a number of other areas that need to be addressed by the next generation of CB-OHSs. For example, creating structures that have endpoints in two different backend servers (5) is still an unsolved problem. Thus, requirements R8 and R12 (consistency across servers) remain open issues. Furthermore, as mentioned in Section 3, the OHSWG has not standardized the interface between middleware services and backend data stores. Therefore, interoperability of these components has not yet been achieved by the OHSWG. Achieving this type of interoperability would potentially ease the development of new middleware components. The potential for this approach was demonstrated at Hypertext'99 by the CAOS application (54) that utilized a navigational service and a spatial hypermedia service built on top of a common backend data store. This example backs the requirements R5 and R6, i.e., the support of multiple structures and the means of combining them. This approach allowed navigational links into and out of a spatial hypermedia workspace, since each middleware component understood the structural abstractions of the other.

Following this thought, we might ask the question where the commonalities between different domains end. Clearly, a common storage interface would mainly support manipulation of structures (also referred to as "structural computing" (32)). The difficulties here lie in a common definition or abstraction of structure both in terms of the data model and its behavior such that the notion of structure can be supported by e.g. operating systems, programming languages or design models. However, this would provide a basis for addressing the requirements R8 and R12.

5.3 Low-level Improvements to the Open Hypermedia Framework

Following the discussion of interfaces described above, we will now address two low-level areas of the framework that could benefit with alternative approaches to initial design decisions.

5.3.1 Streaming

A somewhat different, but nevertheless pressing aspect of the OHSWG's interoperability efforts concerns the streaming of data. Many application areas demand the streaming of multimedia content, e.g. broadcasting of audio or video. While at first glance, this issue could seemingly be addressed by an improvement to the underlying communication protocol of the open hypermedia protocol, the domain of hypermedia adds a difficult issue related to the synchronization of link data with content data (60). One open issue is how can anchors be "summarized" in case frames of a video broadcast have to be dropped because of varying bandwidth. Additionally, streaming requires a different underlying processing model. Requests such as *GetAllEndpoints* cannot be applied to a stream of content, since the content will only exist at some future point in time. Hence applications have to be designed to accommodate a more asynchronous nature of communicating with CB-OHSs.

5.3.2 Communication Mechanisms

With respect to the mechanisms used for communicating between components of the open hypermedia framework, the OHSWG takes the approach that ideally the definition of the interfaces should be independent of the actual on-the-wire protocol. Hence, the OHSWG has sponsored experiments using Java's Remote Method Invocation mechanism for communication as well as using CORBA (37). For true interoperability however, it is necessary to agree on the physical protocol and message format communicated; for the demonstrations mentioned above (see Section 4.4) the OHSWG relied on a message format encoded in XML and communicated via plain TCP/IP sockets. Anderson et al. (38) identify additional communication-related issues such as security, encoding, and stateless vs. state-preserving protocols. These issues have yet to be addressed by the OHSWG.

6 CONCLUSIONS

The Open Hypermedia Systems Working Group (OHSWG) has achieved a great deal of success. The effort began with the goal of achieving client interoperability of open hypermedia systems. The OHSWG demonstration at Hypertext'98 represented the first real milestone towards completing that goal: A general navigational interface (OHP-Nav) has been designed and developed and two open hypermedia systems have implemented the interface. The next step is to promote the adoption of the interface by additional open hypermedia systems.

However, much more than the original goal has been achieved. The OHSWG has been successful in generating an active research community in the domain of open hypermedia systems that has addressed topics outside the initial navigational domain, such as client-side services, link traversal behaviors, open hypermedia reference architectures, and support for spatial hypermedia. The workshops held by the community served to capture the evolution of the design rationale behind the open hypermedia framework. Finally, the community is starting to build on the infrastructure produced so far to spur the development of new protocols and new technology that will serve as building blocks for more powerful clients providing access to pervasive hypermedia services. These last two points deserve further elaboration.

6.1 Evolution of Design Rationale

The best property of the design of the open hypermedia framework is that it *evolved*. The authors of the initial protocol did not attempt to force a particular design on the open hypermedia community. Rather they stated a desirable goal (client interoperability for open hypermedia systems) and one possible approach to achieving it and presented the work to the community for feedback. Similar to the standards process of the IETF, this approach allowed other members of the open hypermedia community to become stakeholders in the protocol and to commit to achieving the stated goal. The initial protocol underwent substantial revision after two workshops consisting of informal discussion and a formal critique (38). The protocol was critiqued along syntactic and semantic dimensions, with the latter raising serious issues with respect to security, protocol design, service definition, and user-interface prescriptions (38). The second revision of the protocol (a result of the previous feedback and the OHS 3.5 working group meeting) formed the basis for the protocol implemented at the Hypertext'98 demonstration. The key differences from the original were a formal specification of the underlying data model, the adoption of XML as the on-the-wire encoding format, and support for collaborative hypermedia. The navigational protocol is now evolving in ways that further increase its cohesiveness. In particular, issues related to services and collaboration are now being addressed separately by new protocols, as described previously in Section 4.2.

A distinct protocol for services is an excellent example of the evolution of the open hypermedia framework's design. The initial protocol had an operation by which a client could request the names of all the services provided by an open hypermedia system. The protocol specified that OHP-aware clients would display the list of service names in a menu. A user could then invoke one of these services by selecting its associated menu item. This approach to services was criticized along several dimensions. First, the syntax and semantics of services were left unspecified (other than defining a service's name). It was thus unclear what information a service required and what information it would return once invoked. As a result, a client would require built in knowledge in order to access a particular service that might only be offered by a single open hypermedia system. It was argued that it would be better to standardize the set of operations required by the protocol to support the navigational domain and to leave unspecified services outside the domain of the protocol. Second, the fact that the protocol specified a particular user-interface for accessing the services was considered too restrictive. It was argued that a low-level protocol should not contain any prescriptions on a client's user-interface. The result of these arguments was that indeed the services required for the navigational domain were standardized; however a vestige of the original service mechanism remained.

The process of standardizing the navigational protocol however led to the component-based point-of-view adopted by the OHSWG at the OHS 3.5 working group meeting in September 1997 in Århus, Denmark. This point-of-view essentially viewed the efforts of the OHSWG as producing an open hypermedia framework of which the navigational domain would be supported by one protocol. This would allow other protocols to be added to the framework which would address other domains such as spatial and collaborative hypermedia. It was this fundamental shift in perspective that ultimately led to the final solution to the notion of services specified by the original protocol. First, all notions of externally defined services were removed from the navigational protocol and placed in the service described in Section 4.2.3. This removal increases the cohesiveness of the navigational protocol since it is no longer saddled with operations unrelated to enabling hypermedia navigation. As expressed by Antoine de Saint-Exupéry, *"Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away."* The OHSWG is nearing this point with the design of the navigational protocol and has begun the process of evolution on several new protocols that promise to increase the power and sophistication of the hypermedia functionality that can be provided to future clients.

6.2 Empowering Future Applications

The open hypermedia framework now consists of an evolving set of protocols that are standardizing access to the hypermedia functionality of multiple hypermedia domains. For instance, consider the CAOS spatial hypermedia client demonstrated at the Hypertext'99 conference in Darmstadt, Germany and developed by a research group at Aarhus University (54). The CAOS client makes use of three open hypermedia interfaces in its implementation: the navigational interface, the evolving spatial hypermedia interface, and the newly defined session management interface (discussed in Section 4.2). The first interface allows objects in the CAOS client to link to other objects managed by any other OHP-enabled application. Even more important, the CAOS client can access its navigational services from any OHP-enabled open hypermedia system, not just the Construct server upon which it was developed. The second interface serves to prototype the services required by a standardized interface for spatial hypermedia and again prepares the CAOS client for receiving its spatial hypermedia services from other hypermedia systems in the future. Finally, the third interface allows the user of the CAOS client to participate in collaborative sessions with other users. This service works in tandem with the first interface to provide functionality such as having a user perform a link traversal for all of the users in his collaborative session.

The significance of this client cannot be overstated. Previous to this achievement, no hypermedia research group had produced an application that provided collaborative access to more than one domain of hypermedia functionality. The existence of standardized hypermedia services raises the level of abstraction at which developers work when designing and implementing hypermedia functionality within their clients. Previously, integrating open hypermedia into a client required a developer to pick a particular open hypermedia system and be tied to its non-standard interface for hypermedia services. Now, developers can integrate open hypermedia services into their applications and feel confident that their users can access those hypermedia services regardless of the actual open hypermedia system providing them. The pioneers of the hypermedia field, Bush, Nelson, and Engelbart, all had as part of their core vision the notion that hypermedia services should be provided to users pervasively throughout their computing environment. The work of the OHSWG has provided a solid foundation upon which this vision of pervasive access can be achieved.

7 ON-LINE RESOURCES

The Open Hypermedia Systems Working Group pages are available as <http://www.ohswg.org/>.

The OHS 1 Workshop pages can be retrieved from <http://www.aue.auc.dk/~kock/OHS-ECHT94/>.

The OHS 2 Workshop pages can be obtained from <http://www.aue.auc.dk/~kock/OHS-HT96/>.

The OHS 3 Workshop pages can be retrieved from <http://www.aue.auc.dk/~kock/OHS-HT97/>.

The OHS 4 Workshop pages are available as <http://www.aue.auc.dk/~kock/OHS-HT98/>.

The OHS 5 Workshop pages can be retrieved from <http://www.aue.auc.dk/~kock/OHS-HT99/>.

The JoDI (Journal of Digital Information) Special Issue on Open Hypermedia can be obtained from <http://jodi.ecs.soton.ac.uk/View/Index/v01/i02/>.

Information on HyperWave can be obtained from <http://www.hyperwave.com/>.

The Microcosm home page is available as <http://www.multicosm.com/>.

The home page of the Coconut project is <http://www.cit.dk/coconut/>.

8 ACKNOWLEDGMENTS

The authors would like to thank the OHS community, all of whom have contributed to this paper. Special thanks are due to Dirk Bucka-Lassen, Jon Griffiths, Wendy Hall, Ian Heath, Claus Ågaard Pederson, Kasper Østerbye, Olav Reinert, Antoine Rizk, Lloyd Rutledge, Lennert Sloth, Richard Taylor, Randy Trigg and Jim Whitehead.

9 REFERENCES

(1) MOLINE, J., BENIGNI, D., and BARONAS, J., editors (1990). *Proceedings of the Hypertext Standardization Workshop, January 16-18, 1990*, National Institute of Standards and Technology (NIST) Special Publication 500-178, Gaithersburg, MD 20899.

(2) ANDERSON, K. M., TAYLOR, R. N., and WHITEHEAD, E. J. Chimera: Hypertext for heterogeneous software environments. In *ECHT '94. Proceedings of the ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK* (1994), pp. 94-197.

(3) GRØNBÆK, K., BOUVIN, N. O., and SLOTH, L. Designing Dexter-based hypermedia services for the world wide web. In *Proceedings of the eighth ACM conference on Hypertext, April 6-11, 1997, Southampton, UK* (1997), pp. 146-156.

(4) NÜRNBERG, P. J., LEGGETT, J. J., and SCHNEIDER, E. R. Hypermedia operating systems: A new paradigm for computing. In *Proceedings of the seventh ACM conference on Hypertext '96, March 16-20, 1996, Washington, D.C.* (1996), pp. 194-202.

(5) WIIL, U. K., and LEGGETT, J. J. Workspaces: The HyperDisco approach to Internet distribution In *Proceedings of the eighth ACM conference on Hypertext, April 6-11, 1997, Southampton, UK* (1997), pp. 13-23.

(6) ANDREWS, K., KAPPE, F., and MAURER, H. Serving information to the web with Hyper-G. *The Third International World-Wide Web Conference. Darmstadt, Germany 27* (1995), 919-926. Published in Computer Networks and ISDN Systems.

(7) DAVIS, H., HALL, W., HEATH, I., HILL, G., and WILKINS, R. Towards an integrated information environment with open hypermedia systems. In *ECHT '92. Proceedings of the ACM conference on Hypertext, November 30-December 4, 1992, Milan, Italy* (1992), pp. 181-190.

(8) PEARL, A. Sun's link service: A protocol for open linking. In *Second annual ACM conference proceedings on Hypertext '89, Nov. 5-9, 1989, Pittsburgh, PA* (1989), pp. 137-146.

(9) NÜRNBERG, P. J., LEGGETT, J. J., and WIIL, U. K. An agenda for open hypermedia research. In *Proceedings of the ninth ACM conference on Hypertext and hypermedia, June 20-24, 1998, Pittsburgh, PA* (1998), pp. 198-206.

(10) AßFALG, R. (ed.). *The Proceedings of the Workshop on Open Hypertext Systems, Konstanz*, May 1994.

(11) WIIL, U. K., and ØSTERBYE, K., Eds. *Proceedings of the ECHT '94 Workshop on Open Hypermedia Systems.* Technical Report R-94-2038, Dept. of Computer Science, Aalborg University.

(12) WIIL, U. K., and DEMEYER, S., Eds. *Proceedings of the 2nd Workshop on Open Hypermedia Systems, Hypertext '96, Washington, D.C., March 16-20. Available as Technical Report No. ICS-TR-96-10 from the Department of Information and Computer Science, University of California, Irvine* (1996).

(13) WIIL, U. K., Ed. *Proceedings of the 3rd Workshop on Open Hypermedia Systems, Hypertext '97, Southampton, England, April 6-11* (1997). Scientific Report 97-01, The Danish National Centre for IT Research.

(14) WIIL, U. K., Ed. *Proceedings of the 4th Workshop on Open Hypermedia Systems, Hypertext '98, Pittsburgh, PA, June 20-24* (1998). Technical Report CS-98-01, Aalborg University Esbjerg, DK.

(15) WIIL, U. K., Ed. *Proceedings of the 5th Workshop on Open Hypermedia Systems, Hypertext '99, Darmstadt, Germany, February 21-25* (1999). Technical Report CS-99-01, Aalborg University Esbjerg, DK.

(16) DAVIS, H. C., RIZK, A., and LEWIS, A. OHP: A draft proposal for a standard open hypermedia protocol. In *Proceedings of the 2nd Workshop on Open Hypermedia Systems, Hypertext '96, Washington, D.C., March 16-20. Available as Technical Report No. ICS-TR-96-10 from the Department of Information and Computer Science, University of California, Irvine* (1996), U. K. Wiil and S. Demeyer, Eds., pp. 27-53.

(17) RIZK, A. and SAUTER, L. Multicard: An open hypermedia system. In *ECHT '92. Proceedings of the ACM conference on Hypertext, November 30-December 4, 1992, Milan, Italy* (1992), pp. 4-10.

(18) DAVIS, H. C., KNIGHT, S., and HALL, W. Light hypermedia link services: A study of third party application integration. In *ECHT '94. Proceedings of the ACM European conference on Hypermedia technology, Sept. 18-23, 1994, Edinburgh, Scotland, UK* (1994), pp. 41-50.

(19) MARSHALL, C. C., and SHIPMAN, F. M. Spatial hypertext: Designing for change. *Communications of the ACM 38, 8* (1995), 88-97.

(20) van PARUNAK, H. Don't link me in: Set-based hypermedia for taxonomic reasoning. In *Third annual ACM conference proceedings on Hypertext '91, Dec. 15-18, 1991, San Antonio, TX* (1991), pp. 233-242.

(21) NELSON, T. H. *Literary Machines*. Published by the author. Distributed by Eastgate Systems, 1981.

(22) ØSTERBYE, K., and WIIL, U. K. The flag taxonomy of open hypermedia systems. In *Proceedings of the seventh ACM conference on Hypertext '96, March 16-20, 1996, Washington, D.C.* (1996), pp. 129-139.

(23) NÜRNBERG, P. J., and ASHMAN, H. What was the question? Reconciling open hypermedia and World Wide Web research. In *Proceedings of the '99 ACM conference on Hypertext, February 21-25, 1999, Darmstadt, Germany* (Feb. 1999), pp. 83-90.

(24) BOUVIN, N. O. Unifying strategies for Web augmentation. In *Proceedings of the '99 ACM conference on Hypertext, February 21-25, 1999, Darmstadt, Germany* (Feb. 1999), pp. 91-100.

(25) WHITEHEAD, J. E. An architectural model for application integration in open hypermedia environments. In *Proceedings of the '97 ACM conference on Hypertext, April 6-11, 1997, Southampton, UK* (1997), pp. 1-12.

(26) BUSH, V. 1945. As we may think. *The Atlantic Monthly 176*(1) (1945), 101-108.

(27) ØSTERBYE, K. Fred the programmer. In *Proceedings of the 3rd Workshop on Open Hypermedia Systems, Hypertext '97, Southampton, UK, April 6-11. Available as Technical Report No. SR-97-01 from the Danish National Centre for IT Research, 8000 Aarhus C, Denmark* (1997), pp. 146-149.

(28) DOURISH, P., and BELOTTI, V. Awareness and Coordination in Shared Workspaces. In *Proceedings CSCW'92, October 31-November 4, Toronto,* 1992. ACM Press, pp. 107-114.

(29) TATAR, D. G., FOSTER, G., and BOBROW D. B. Design for Conversation: Lessons from Cognoter. *Int. Journal on Man-Machine Studies 34* (1991), pp. 185-209.

(30) GRØNBÆK, K., and WIIL, U. K. Towards a reference architecture for open hypermedia. *Journal of Digital Information (JoDI). Special Issue on Open Hypermedia Systems 1,* 2 (1997).

(31) NÜRNBERG, P. J., and LEGGETT, J. J. A vision for open hypermedia systems. *Journal of Digital Information (JoDI). Special Issue on Open Hypermedia Systems 1,* 2 (1997).

(32) NÜRNBERG, P., LEGGETT, J., and SCHNEIDER, E. As we should have thought. In *Proceedings of the eighth ACM conference on Hypertext, April 6-11, 1997, Southampton, UK* (1997), pp. 96-101.

(33) GRØNBÆK, K., and TRIGG, R. Toward a Dexter-based model for open hypermedia: Unifying embedded references and link objects. In *Proceedings of the seventh ACM conference on Hypertext '96, March 16-20, 1996, Washington, D.C.* (1996), pp. 149-160.

(34) HALL, W., DAVIS, H. C., and HUTCHINGS, G. *Rethinking Hypermedia: The Microcosm Approach*. Kluwer Academic Publishers, 1996.

(35) HALASZ, F., and SCHWARTZ, M. The Dexter hypertext reference model. *Communications of the ACM 37,* 2 (1994), 30-39.

(36) LEGGETT, J. J., and SCHNASE, J. L. Viewing Dexter with open eyes. *Communications of the ACM 37* (1994), 76-86.

(37) MILLARD, D. E., REICH, S., and DAVIS, H. C. Reworking OHP: the road to OHP-Nav. In *Proceedings of the 4th Workshop on Open Hypermedia Systems, Hypertext '98, Pittsburgh, PA, June 20-24* (June 1998), U. K. Wiil, Ed., pp. 48-53.

(38) ANDERSON, K. M., TAYLOR, R. N., and WHITEHEAD, E. J. A Critique of the Open Hypermedia Protocol. *Journal of Digital Information (JoDI). Special Issue on Open Hypermedia Systems 1,* 2 (1997).

(39) WIIL, U. K., and WHITEHEAD, E. J. Interoperability and open hypermedia systems. In *Proceedings of the 3rd Workshop on Open Hypermedia Systems, ACM Hypertext '97, Southampton, UK, April 6-11. Available as Report No. SR-97-01 from the Danish National Centre for IT Research, 8000 Aarhus C, DK* (1997), pp.137-145.

(40) ENGELBART, D. C. Augmenting human intellect: A conceptual framework. Stanford Research Institute Technical Report AFOSR-3223, Contract AF 49(638)-1024 (Oct. 1962), Palo Alto, CA.

(41) NELSON, T. H. Getting it out of our system. In *Information Retrieval: A Critical View,* (G. Shecter, ed.) Thompson Book Co., Washington, DC, 191-210, 1967.

(42) DAVIS, H. C., MILLARD, D. E., REICH, S., BOUVIN, N. O., GRØNBÆK, K., NÜRNBERG, P. J., SLOTH, L., WIIL, U. K., and ANDERSON, K. M. Interoperability between hypermedia systems: The standardisation work of the OHSWG (technical briefing). In *Proceedings of the '99 ACM conference on Hypertext, February 21-25, 1999, Darmstadt, Germany* (Feb. 1999), pp. 201-202.

(43) WANG, W. and HAAKE, J. Flexible coordination with cooperative hypermedia. In *Proceedings of the ninth ACM conference on Hypertext and hypermedia, June 20-24, 1998, Pittsburgh, PA* (1998), pp. 245-255.

(44) HAAKE, J. M. and WANG, W. Flexible support for business processes: Extending cooperative hypermedia with process support. In *GROUP '97. Proceedings of the international ACM SIGGROUP conference on Supporting group work: the integration challenge, Nov. 16-19, 1997, Phoenix, AZ* (1997), pp. 341-350.

(45) LEWIS, P., DAVIS, H., GRIFFITHS, S., HALL, W., and WILKINS, R. Media-based navigation with generic links. In *Proceedings of the seventh ACM conference on Hypertext '96, March 16-20, 1996, Washington, D.C.* (Mar. 1996), pp. 215-223.

(46) BLACKBURN, S., and DEROURE, D. A tool for content based navigation of music. In *Multimedia '98, Bristol, UK* (Sept. 1998), pp. 361-368.

(47) WIIL, U. K., and NÜRNBERG, P. J. Evolving hypermedia middleware services: Lessons and observations. In *Proceedings of the 1999 ACM Symposium on Applied Computing (SAC '99), San Antonio, TX* (Feb. 1999), pp. 427-436.

(48) GRØNBÆK, K., HEM, J. A., MADSEN, O. L., and SLOTH, L. Designing Dexter-based cooperative hypermedia systems. In *Proceedings of the fifth ACM conference on Hypertext '93, Nov. 14-18, 1993, Seattle, WA* (1993), pp. 25-38.

(49) BECK-WILSON, J., PFISTER, H.-R., SCHUCKMANN, C., and WESSNER, M. (in press). The CLear approach: Designing distributed computer supported cooperative learning environments. In *A. Eurelings (Ed.) Integrating information & communication technology in higher education.* Amsterdam: Kluwer, 1999.

(50) WESSNER, M., BECK-WILSON, J., and PFISTER, H.-H. CLear – a cooperative distributed learning environment. In *Proceedings of ED-MEDIA/ED-TELECOM 98, June 20-25, 1998, Freiburg, Germany*, (1998), pp. 1876-1877.

(51) SCHUCKMANN, C., KIRCHNER, J., SCHÜMMER, L., HAAKE, J. M. Designing object-oriented synchronous groupware with COAST. In *CSCW '96. Proceedings of the ACM 1996 conference on Computer supported cooperative work, Nov. 16-20, 1996, Boston, MA* (1996), pp. 30-38.

(52) GOOSE, S., DALE, J., HILL, G., DEROURE, D., and HALL, W. An open framework for integrating widely distributed hypermedia resources. In *IEEE Conference on Multimedia and Computing Systems, Hiroshima* (June 1996), pp. 364-371.

(53) DEROURE, D., HALL, W., REICH, S., PIKRAKIS, A., HILL, G., and STAIRMAND, M. An open architecture for supporting collaboration on the Web. In *WET ICE '98 - IEEE Seventh International Workshops on Enabling Technologies, June 17-19, Stanford University, California* (1998), pp. 90-95.

(54) REINERT, O., BUCKA-LASSEN, D., PEDERSEN, C. A., AND NÜRNBERG, P. J. CAOS: A collaborative and open spatial structure service component with incremental spatial parsing. In *Proceedings of the '99 ACM conference on Hypertext, February 21-25, 1999, Darmstadt, Germany* (Feb. 1999), pp. 49-50.

(55) CONKLIN, J. H. C., and BEGEMAN, M. L. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems 6*, 4 (1988), 303-331.

(56) WANG, W., and HAAKE, J. M. Implementation issues on OHS-based workflow services. In *Proceedings of the 5th Workshop on Open Hypermedia Systems, ACM Hypertext '99 Conference, Darmstadt, Germany, February 21-25. Available as Report No. CS-99-01 from the Dept. of Computer Science, 6700 Aalborg University Esbjerg, Denmark* (1999), pp. 52-56.

(57) SAWHNEY, N., BALCOM, D., and SMITH, I. HyperCafe: narrative and aesthetic properties of hypervideo. In *Proceedings of the seventh ACM conference on Hypertext '96, March 16-20, 1996, Washington, D.C.* (1996), pp. 1-10

(58) TRIGG, R. H. Guided tours and tabletops: tools for communicating in a hypertext environment. *ACM Transactions on Office Information Systems 6*, 4 (Oct. 1988), 398-414.

(59) ZELLWEGER, P. T. Scripted documents: A hypermedia path mechanism. In *Second annual ACM conference proceedings on Hypertext '89, Nov. 5-9, 1989, Pittsburgh, PA* (1989), pp. 1-14.

(60) DEROURE, D., BLACKBURN, S., OADES, L., READ, J., and RIDGWAY, N. Applying open hypermedia to audio. In *Proceedings of the ninth ACM conference on Hypertext and hypermedia, June 20-24, 1998, Pittsburgh, PA* (1998), pp. 285-286.