REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 1, número 1
Págs. 105-124

The Aristotle Approach to Open Hypermedia

C. Petrou, D. Martakos, M. Hatzopoulos, S. Halamandaris and S. Hadjiefthymiades
Department of Informatics, National & Kapodistrian University of Athens
TYPA Buildings, 15771, Athens, Greece
cpetrou@di.uoa.gr

Abstract

Large-scale distributed hypermedia systems comprise a generation of powerful tools to meet the demands of the new information globalization era. The most promising of such systems have characteristics that allow for the easy adaptation both to an, actually, unpredictable technological evolution and to the constantly evolving information needs of users. Such systems are generally known as Open Hypermedia Systems (OHS). Recently, research effort has been focused on the formulation of a solid set of OHS standards (i.e., protocols, reference models and architectures) that would stem from a common understanding and thus, direct future implementations. Unfortunately, sophisticated hypermedia notions like composite nodes and dynamic linking, still found in older, monolithic hypermedia systems, have been overlooked or roughly portrayed. This paper presents a distributed system architecture implemented on top of a hypermedia model for OHS, which captures notions like these and sets the foundation for straightforward implementations. The proposed hypermedia model comprises a Hypermedia Data Model, which is extensible and fulfils a series of OHS requirements, and a Structural Model, which ensures controlled hyperdocument construction and flexible dynamic linking. Aspects of a prototype hypermedia server (Aristotle), which has been the testbed for this architecture, are discussed.

Keywords: Open Hypermedia Systems, Hypermedia Modeling, Distributed Information Systems

1. Introduction

The hypermedia structural and navigational paradigm is now widely accepted as the cornerstone for distributed information environments. This is primarily manifested by the success of the World-Wide Web (WWW). Undoubtedly, the WWW is the most widely used distributed hypermedia system. Its voluminous data content ceaselessly expands together with the complexity of its link structure. However, the hypermedia model adopted by the WWW is simple as it primarily aims at the construction of widely distributed hypertext and focuses, in a lesser degree on multimedia support, collaboration, link integrity and integration of new media types and external applications. These issues have attracted the interest of researchers. The coordinated efforts towards their resolution have led to the development of Open Hypermedia Systems (OHS). In contrast to OHS, which provide for external, n-ary and bi-directional links, the WWW, based on the HTML standard [1], suffers from a series of shortcomings [2]:

- The dangling link problem: if a node has been physically moved or renamed, or if its content has been altered, it is possible that some link becomes invalid (i.e., cannot be traversed).
- Only the owner of a node can create links departing from the document, while links to specific parts of a node can be made only if the owner has created the relevant destination anchors.
- There is no way to retrieve information about incoming links, as links are strictly uni-directional. In a broader sense, no link sharing and reusing is foreseen.
- It is impossible to define different set of outgoing links on the same node. Consequently, there is no way to implement custom "views" of the hypergraph for different user needs.
- As links are unary and cannot overlap, it is impossible to refer to more than one destination nodes from a single source anchor, or to define more than one anchors in the same location.
- The majority of WWW authors have not exploited link (and node) typing, which promotes understanding of the conceptual relationship between interlinked nodes [3], even though link types are modestly supported in HTML. We claim that the reason for this is the absence of a meta-model, an ontological framework where meta-modelers can define and share conceptual schemata or ontologies. This is also justified by the strong interest that has been shown for WWW meta-data modeling [4].
- Span-to-Span links can only be created between HTML pages and there is no provision for defining anchors in images, audio or video.

The OHS community has agreed on the need for external link storage to avoid such problems. Hence, in most approaches, links are treated as "first-class" hypermedia objects. Historically, the Dexter Reference Model [5] was one of the first widely accepted abstract models that posed the requirement for external link storage. Thereafter, all OHSs conform to this requirement. According to the first OHP proposal [6], the term open implies the possibility of importing new objects into a system. In the case of hypermedia systems this can be further specialized as:

1. Scalability: import every kind of hypermedia objects without limitation, regarding the size or the number of objects that the system can accommodate.
2. Data Formats: support of any data format, including temporal media. As noted in [7], no mark-up upon the data should be imposed which prevents them from being accessible from other, external to OHS, native viewers or editors.
3. Applications: the set of external applications, which have access to hypermedia services, should be open. Nevertheless, the exact process of integrating an external application may vary from

building a specialized client (e.g., Hyper-G Harmony client [8]) to integrating third-party applications (e.g., Microcosm [9]).

4. Data Models: the hypermedia system should not impose a single view on what constitutes a hypermedia data model. Instead, it should promote extensibility and be configurable so that, new data models could be incorporated. This is also referred to as heterogeneity [10] and is a crucial factor in the interoperability between hypermedia systems. Heterogeneity is hard to be achieved as it introduces implications regarding the functional part of the system: in the case of major changes of the data model, the provided services might cease functioning or, at least, they should also be updated. A candidate solution is to de-couple the linking functionality from the rest of the system and "embed" the linking model information into separate linking services. However, this simply shifts the problem to the client-side i.e., special "thick" clients should be available for every different linking service to handle both different linking models and different media types. Another, more compromising direction, is to de-couple media specific details from pure hypermedia concepts. The advantage is that the clients remain "thin", as they need to be aware of a single model. This issue is more complicated as there is no consensus within the OHS Research Group (OHSRG) on which services constitute the "core" of an OHS [32]. In any case, well-known notions like anchors, links, nodes, composites, and presentation specifications (pspecs) should be captured [33].
5. Distribution: the system and the integrated services, application and data stores must be capable of being distributed across different platforms and network locations.
6. Users: the system must support multiple concurrent users, cater for concurrency control, access permissions and allow users to maintain their own private view of the hypermedia objects and structures. As noted in [3], collaboration support should also be provided.
7. Computation: the system should also provide dynamic and virtual structures, whose construction is managed either automatically (e.g., generic links in Microcosm), or performed by the user (e.g., structure-based or content-based search queries).

The issue of hypermedia content storage has driven research/implementation efforts towards two directions [10]. The first, described as the "Link Service Systems" (LSS) approach, focuses on the provision of hypermedia functionality to client applications, orthogonally to display and storage functionality [11]. Systems like DLS, Chimera and Microcosm fall into this category. The second direction has been followed by "Hyperbase Management Systems" (HBMS) and, apart from the hypermedia services, also ensures content storage on integrated or external distributed data sources. Examples from this category are DHM, HyperDisco and Hyper-G. We should stress that, beyond the differences of these approaches, there is a common need to store and manipulate hypermedia objects and structures, as for example the demand for external (to content) storage of links depicts. The gap between these two approaches is further narrowing as the mutual influence has led HBMS to be deployed as part of open, extensible and distributed architectures and LSS to incorporate storage, collaboration and versioning facilities [11].

The fulfillment of the aforementioned requirements is not a trivial task. Examining WWW against these requirements reveals that only "Scalability" and "Distribution" are fully satisfied, while part of the "Users" requirement is also met. To provide an integrated solution, this paper suggests the coupling between OHS principles and existing/upcoming WWW standards (e.g., XML). This architecture has been based on a generic data model for OHS, which captures notions like composites and transclusion [24]. The core entity of this architecture is the Aristotle hypermedia server. The suggested hypermedia model decomposes into a Hypermedia Data Model, which fulfills OHS requirements, and a Structural

Model, which ensures controlled hyperdocument construction and flexible dynamic linking. As the overall architecture complies with the "Applications" requirement, a typical WWW browser is treated as another client application that may participate in hypermedia functionality and retain its native format (HTML) and transport mechanism (HTTP).

The paper is organized as follows. Section 2 presents the Aristotle server system architecture and its components. Section 3 discusses the underlying model. Section 4 evaluates our approach against OHS taxonomies and abstract models. Finally, Section 5 presents our conclusions and future research directions.

2. System architecture

The overall system architecture is presented in \h Figure 1. Aristotle servers have a modular architecture, which facilitates upgrade and promotes a clear decomposition of complex procedures to simple request/reply dialogs between modules and through well-defined interfaces.

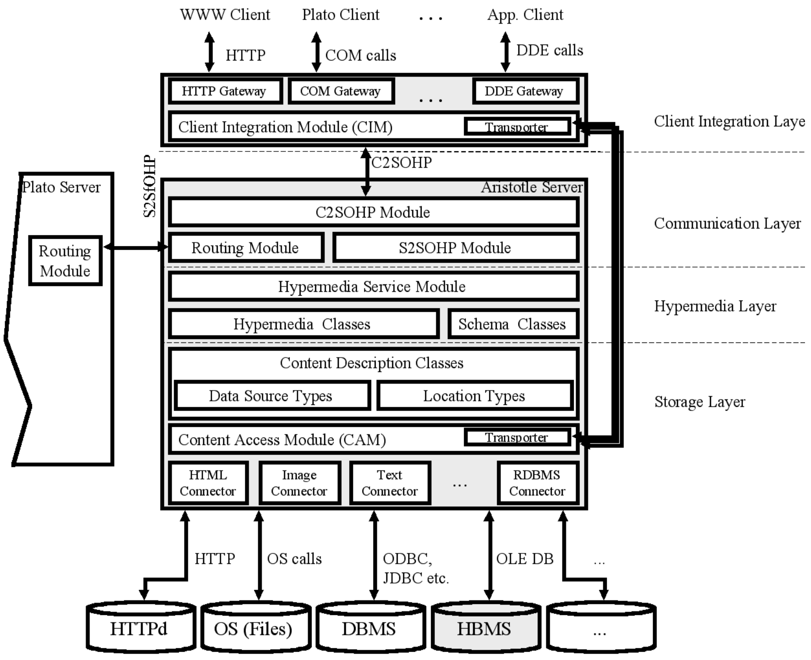

Figure 1: Aristotle Server system architecture

Every Aristotle server stores hypermedia information (but not content) into an HBMS, which in the case of our prototype, is a RDBMS (Microsoft SQL Server). In general, the server is capable of using any RDBMS that has some connectivity interface. Inter-server communication is based on the Server to Server Open Hypermedia Protocol (S2SOHP). Client applications communicate with Aristotle servers through the Client to Server Open Hypermedia Protocol (C2SOHP). Both protocols have been formally defined as XML DTDs. Hence, encoding of messages produces valid XML documents. In the next paragraphs we discuss in detail the modules of the proposed architecture, their functional role and their interaction.

2.1 Clients

In recent bibliography various approaches for integrating third-party applications in an OHS have been proposed. Most of these introduce some “integrator” or “wrapper” entity, which enhances applications with hypermedia capabilities. In the Chimera system [12] this entity has been implemented as a programming library for the UNIX environment. The Devise system [13] employs DDE, OLE and HTTP to communicate with client applications. Analogously, the WebVise system [2] is based on COM for the integration of MS-Office applications and uses a combination of HTTP/ proxy in the case of WWW browsers. The HyperDisco [10] has demonstrated the integration of Emacs and XEmacs [11] editors through macro programming. In another well-known OHS, the HyperG/ Hyperwave [8], a set of custom clients has been developed (Harmony, Amadeus). Since these clients are intrinsically “hypermedia-aware”, they directly communicate with HyperG servers over TCP. In addition, HyperG employs the technique of a pseudo-httpd to integrate regular WWW browsers. The Microcosm ([7], [14]) was one of the first OHS to demonstrate the integration of third-party applications, with various “degrees”, using DDE and MS-Windows clipboard. Microcosm’s evolution led to the Microcosm TNG ([15], [16]) and accompanied with an essential decoupling from specific communication techniques: a set of special entities, like the Daemon, Router, Process Manager etc., took the responsibility for client integration and communication. We conclude this brief review with the DLS/ Agent DLS approach ([17] and [18]), which, similarly to HyperG, employs HTTP dialogs to integrate WWW browsers.

Most of the aforementioned approaches exhibit a certain inflexibility to re-use communication components. For example, consider the case of two applications that are capable of communicating DDE calls. It would be possible for a single “DDE gateway” component to establish DDE connections simultaneously with both applications and manage the flow of DDE requests/ replies. This type of “communication” re-use influenced the design of the Client Integration Layer of Aristotle.

The core entity of this layer is the Client Integration Module (CIM) service, which communicates directly with Aristotle servers via the C2SOHP. For every integrated application, a local registry stores information about the hypermedia objects it can handle and the services it offers, along with XML-encoded, communication-depended service invocation instructions (termed Service Invocation Script, SIS). Conceptually, a registry entry is a tupple:

Media Type, Location Addressing, Hypermedia Object, Service on the Hypermedia Object, SIS

The actual definition of such a tupple is as follows:

<DST, DLT, HOT, SN, SIS>

where the DST refers to the Data Source Type and DLT to the Data Location Type that describe the media type and the addressing scheme for media content, the Hypermedia Object Type (HOT) refers to the hypermedia data model object class for which the service is offered and the Service Name (SN) is the unique identifier of the hypermedia service. DST, DLT and HOT will be explained shortly in section 3.

When such a service is invoked (e.g., “present a text node”), the CIM service consults the registry, locates the appropriate “gateway” and spawns an instance of it. After the activation of the “gateway”, the CIM service forwards the service request in the form of an XML-encoded SIS. The “gateway” uses the native communication mechanism of the application (e.g., DDE, OLE etc.) to forward the request and accept any response, which will then be forwarded back to the CIM service.

We claim that this architecture promotes component re-use, simplifies client integration and increases expandability. The re-use advantage is rather obvious, considering that the same set of “gateways” may support communication with a superset of applications, a feature that is missing from “wrapper”-oriented approaches. Clients are not exposed to C2SOHP thus they may be easily integrated since they need not incorporate XML parsing/ composing capabilities. Due to the fact that the SIS encodes services, altering a service in most cases simply means to update the SIS rather to re-program the application. Finally, the set of “gateways” is interminable expandable and new “gateways” may be added even at run-time. This last feature is feasible since the “gateway”-to-CIM service communication is based on dynamic objects (COM), object late binding and a solid “gateway”-to-CIM protocol.

In pursuit of greater performance, the decoupling of control messages and media content flow was adopted. Control messages implement the C2SOHP and bi-directional flow from CIM service to “gateways”. A special part of the CIM, termed Transporter, undertakes the transfer of the actual media content (e.g., the string content of a “text” node, the binary content of an “image” node etc.). This multi-threaded service dynamically allocates direct socket connections with its peer entity of the Aristotle server(s) that will provide or should receive the media content of some node.

A special type of clients is the Aristotle Client. These are external applications, specifically implemented for communicating with Aristotle Servers and capable of presenting a specific type of media content. A wide range of Aristotle Clients have been developed (TextClient, HTMLClient, ImageClient, VideoClient and SoundClient). All these clients achieve maximum hypermedia functionality (e.g. anchor creation, link traversal etc.) according to the underlying Data Model.

2.2 Client to Server Open Hypermedia Protocol (C2SOHP) Module

This module implements the C2SOHP protocol (not presented here) that is employed for the communication with client applications. The C2SOHP messages are encoded in XML [19] and the protocol has been defined as an XML DTD. Hence, the discussed module undertakes the role of an XML parser/ composer. XML was preferred for the protocol encoding as it is straightforwardly usable over the Internet. It is also easy to create and process XML documents while the language inherits attractive features of the SGML without the relevant complexity. This module also handles network communication with CIM.

2.3 Server-to-Server Open Hypermedia Protocol (S2SOHP) Module

This module implements the S2SOHP protocol (not discussed in this paper) that is employed for inter-Server communication. Similarly to the C2SOHP, S2SOHP messages are encoded in XML.

The module encompasses XML parsing/ composing functionality but does not undertake any network communication as the Routing Module provides such services. Since the formal definition of the protocol is locally stored (as a DTD), it is possible to update/ expand S2SOHP simply by updating the relevant DTD (the same also holds for the C2SOHP).

2.4 Routing Module (RM)

In a large-scale distributed environment, no assumptions can be made about the total number of servers. Maintaining an exhaustive list of addressing information is considered a troublesome approach. As an alternative, we have developed a routing algorithm based on server clusters, random transmission of messages (flooding, analogously to [20]) and message signatures. The algorithm supports uni-cast, multi-cast and broadcast transmission and ensures message delivery. The Routing Module receives messages from the S2SOHP Module and routes them to the appropriate peer Module of the receiving Server(s) using this routing algorithm.

2.5 Hypermedia Service Module (HSM)

HSM is the core module, which coordinates other modules and implements the program logic of the hypermedia behavior as it is specified in the Hypermedia & Schema Classes (discussed in subsequent paragraphs). The translation of C2SOHP to S2SOHP messages (and vice versa) is also a function of the HSM. With the proposed distinction between hypermedia and content specific information (discussed in Section 3), the HSM remains independent from media-specific details; thus, adding support for a new media types does not affect its operation. The HSM has been formally defined using the Vienna Design Methodology (VDM-SL) and is neutral to specific implementations.

2.6 Hypermedia & Schema Classes

Hypermedia Classes is a set of object classes that implement the proposed Hypermedia Data Model. Analogously, Schema Classes implement the Hypermedia Structural Model. A detailed discussion of Hypermedia Classes can be found in Section 3. The Hypermedia Structural Model comprises a visual notation and formal specifications for the definition of hypermedia schemata. If such a schema is applied to a hyperdocument, it ensures the semi-structured and controlled expansion of the hyperdocument. Extensive discussion on schema-based hyperdocuments can be found in [21] and [22].

2.7 Content Description Classes

The purpose of Content Description Classes is to model media-specific meta-information, which is required to (physically) locate a media component or (logically) a part of a component. Two distinct but interrelated object specialization hierarchies exist:

(a) Data Source Type (DST) Hierarchy: In principle, the node elements of this hierarchy correspond to discrete, in terms of media type, data sources. Each element incorporates all the necessary properties to physically locate the media content, along with the declaration of the access method to be employed. This hierarchy may expand; hence new media types could easily be integrated.

(b) Data Location Type (DLT) Hierarchy: The elements of this hierarchy provide the means to logically describe parts of a specific Data Source Type. The description of a part can be static or dynamic. Intuitively, in the first case, the attributes of the DST are interpreted as coordinates in some n-dimensional space where the content of a certain type of Data Source may be addressed, resembling a HyTime-like approach. For instance, this space is:

- 1-dimensional for text and a start/ length pair of numbers defines a part of the text,
- 2-dimensional for image thus, two pairs for coordinates are required and
- 3-dimensional for video (2-dimensional for defining the “shape” anchor, plus one dimension to define time in the sense of a list of frame numbers), etc.

The definition of a dynamic location (e.g., a query in a relational DBMS) is accomplished through some script, which is media-specific and interpreted by a specified Connector of the Content Access Module (discussed in subsequent paragraph). Even though the script is not interpretable by the HSM, the latter module has access to a Script Interface Definition (ScID), having the following structure:

- Name: A text giving a short description of the Script.
- Description: A more analytical description of the Script.
- InParameters: A list of tuples [PName, Type, Size, Value] where the PName is the unique identifier of the parameter across the Script Interface Definition, Type gets a value from a predefined set of supported data types, Size defines the size (in bytes) for the parameter and, finally, Value stores the current parameter value (e.g., [“Start”, INTEGER, 2] defines an integer-valued parameter with name “Start”).
- OutParameters: Similarly to InParameters, this list describes output parameters.

The proposed system initially defines the specialization hierarchies shown in \h Figure 2, but both hierarchies can be expanded in order to incorporate new media types.

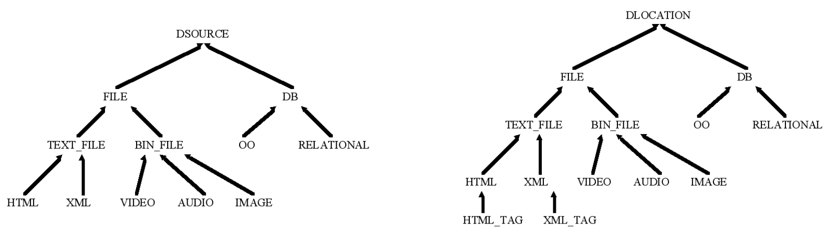

Figure 2: Data Source Type and Data Location Type object hierarchies

Two remarks should be made. Firstly, for each DST at least one corresponding DLT must exist. This ensures that parts of the content of a specific DST can be retrieved. If the DST is related with more that one DLTs then more that one media-dependent part definitions are provided. Secondly, the content of a certain DST is addressable by the corresponding DLT, and also by the immediate and the recursive predecessors of the DLT. As an example, consider a TEXT_FILE DST instance, which can be accessed as a whole, with a corresponding FILE DLT instance, or only a part of it can be retrieved using the corresponding TEXT_FILE DLT instance.

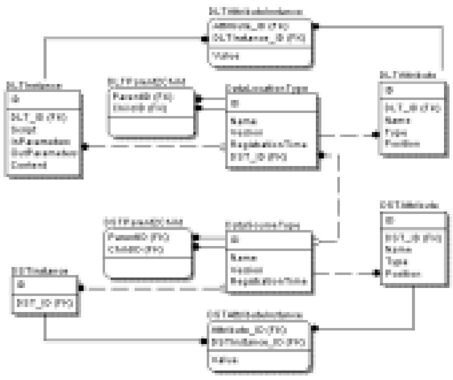

Figure 3: Entity-Relationship diagram of the adopted relational modeling for Data Source Types and Data Location Types

The type hierarchies is partly equivalent to the “Within-Component Layer” of the Dexter Reference Model, as they do not carry hypermedia semantics but only media descriptive information. In order to ensure the expandability of the two hierarchies, these have been modeled as relational tables. All the relevant information is kept in the local Hyperbase. The relevant Entity-Relationship diagram, is presented in \h Figure 3 using the IDEF1x [23] notation. Following this modeling approach, the addition of support for a new media type is achieved in four steps:

1. The server receives a C2SOHP request by some client for the addition of a new DST.
2. The HSM updates the relevant tables of the local Hyperbase.
3. Then, the DTDs of the C2SOHP and S2SOHP protocols are automatically expanded to include element and attribute definitions for the newly added data type. The conversion of a new DST to XML DTD elements, follows the rules given below:
    - A new XML element is created.
    - All DST properties that occur once are converted to XML attributes of the mentioned element.
    - All DST properties that may have multiple occurrences are converted to XML elements and then added to the definition of the first XML element.
4. Possibly, a new Connector (see Section 2.8) is transferred and added to the Content Access Module or one of the available Connectors is re-used to ensure media access.

The strong advantage of this mechanism is that support for a new media type can be introduced, not only by avoiding re-programming some part of the server, but even without disrupting the normal operation of the server. Furthermore, this reconfiguration can be made remotely (through C2SOHP and under authentication control), thus maximizing server’s capability to support new media almost instantly and minimizing administrative cost.

2.8 Content Access Module (CAM)

The Content Access Module comprises the CAM service and a collection of autonomous components, called Connectors. The crucial requirements for “Data formats” and “Scalability” are best achieved if the system remains transparent to data formats and access details. The role of the Connector entity is to ensure such transparency by "gluing" the rest of Aristotle architecture with an open-ended set of data sources (e.g., OS files, RDBMSs etc.). Every Connector implements three basic functions, READ_LOC, WRITE_LOC, REMOVE_LOC. Each function takes as arguments, information about the client who made the request, a DST and one or more DSL instances. Additionally, two more functions may be supported, CONNECT_DS and DISCONNECT_DS with only client information and a DST instance as arguments. The application of location composition (discussed in section 3) remains under the responsibility of Connectors. In the current state of our prototype, Connectors for text, image, sound and video files, as well as an RDBMS Connector have been developed.

The CAM service maintains registration information about Connectors in order to be able to communicate with them. Every Connector is related (bound) with one or more DSTs. The CAM service acts as a mediator, i.e., it makes use of relevant registration and binding information in order to serve an incoming (HSM originated) request for data. As soon as a Connector decodes function call parameters and performs data read/writw operation, the Transporter element undertakes communication control. This element establishes a direct Socket connection with its peer entity of the client-side in order to transfer the requested data. Thereby, communication bottlenecks are avoided and data transfer is more efficient since it is distributed to many threads of the Transporter and is managed in parallel.

In our prototype, CAM entities communicate trough a message queuing mechanism (MSMQ). Initial experimental results have shown that CAM performance ranges from 970 requests/min to 750 requests/min, depending on the type of requested data and the number of location compositions (see Section 3).

3. Hypermedia Data Model

The proposed model is an abstract open hypermedia model comprising six basic classes. For every class we present its properties. The value of a property can be system provided, user provided or masked. A masked property is a production rule whose value can be computed from other property values. An example of a masked property is the unique object identifier (UID). Furthermore, some properties after been given a value, retain this initial value for the rest of the instance's life. This type of properties is termed write-once (w-o) to distinguish from the usual (write-many) properties.

3.1 Data Sources (DS):

A Data Source is an abstract description of some data store (e.g., a file) or supplier (e.g., a DBMS). A DS instance includes information about where to find the data that makes up its content and how to access them. The definition of a DS has as follows:

- UID: Unique object identifier across hypermedia (masked).
- MigrationUID: The new UID of the object, in case the DS instance has been moved in the domain of another server. (w-o, system provided)
- ServerID: The unique identifier of the server that has access to the DS. We note that more than one server can refer more than once to the same DS (w-o).
- LocalID: The local identifier of the DS in the scope of the server ServerID (w-o, system provided).
- SourceType: A property, whose value is an instance node of the Data Source Type hierarchy. This

instance node describes the type of the DS and restricts the location definitions that can be, subsequently, defined upon the DS. Depending on the DST definition, some properties of the instance may be re-writable while others write-only.

- Content: The media content of the DS.

Upon creation of a new DS instance, a new DL is automatically defined its location type covers the whole content of the DS.

3.2 Data Locations (DL):

A Data Location stores all the necessary information to locate a part on the contents of a DS. This class resembles the dataRef entity defined in the Navigation Domain Model proposed by the OHSWG [14], but its role is broader than just to define the persistent location for an anchor. Three main enhancements are provided. Firstly, a DL serves as the building block of the nodes. Secondly, it is capable of defining, both statically and dynamically, locations and finally, apart from referring to a DS, it is possible to refer to another DL thus, providing location abstraction. The properties of the class are:

- UID: Unique object identifier across hypermedia (masked).
- ServerID: The unique identifier of the server that has access to the DL instance (w-o).
- LocalID: The local identifier of the DL in the scope of the server ServerID (w-o, system provided).
- ReferenceID: The unique identifier of the DS or another DL on which the DL defines a new location (w-o).
- LocationType: The value of this property is an instance node of the Data Location Type hierarchy. Depending on the DLT definition, some properties of the instance may be re-writable where others write-only. LocationType can be left empty iff ReferenceID is not empty.
- Content: The media content of the DL.

Different combinations of property values for ReferenceID and LocationType provide three different types of location abstractions. \h Figure 4 illustrates these types:

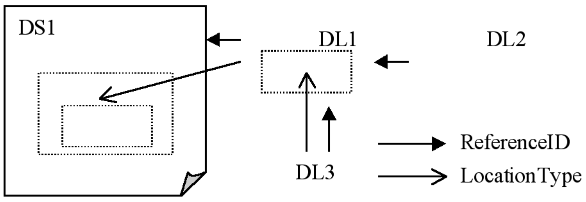

Figure 4: The three types of location abstraction that Data Location objects provide

In \h Figure 4, Data Source DS1 may be of any source type (e.g., a text file). Data Location DL1 has been defined with the ReferenceID pointing to the DS1 and its LocationType includes all the necessary information to define a part into the contents of DS1 (e.g., start and length values). This type of absolute location description is rather straightforward and can be found in the data models of the majority of OHSs. Unlike DL1, Data Locations DS2 and DS3 depict a more elaborated type of location abstraction and form the basis for the definition of complex entities like composite and abstract nodes. Data Location DL2 only holds a ReferenceID to DL1 and has an empty LocationType. Its use is ideal in cases where we like to construct an abstract Node which “points” to locations that have not yet been “frozen” (i.e., DL1 marks a certain section in the DS1 while the editing of DS1 is in-progress). In such cases, only the LocationType of the first DL has to be updated (i.e., DL1), without any updates to referrer DLs (i.e., DL2). The last type of location abstraction is exhibited in DL3, which holds a ReferenceID to DL1 and defines a new, relevant to DL1s LocationType. The obvious advantage is that the LocationType of DL3 remains valid even if the LocationType of DL1 gets invalid (i.e., in DS1 some text has been inserted before the LocationType of DL1).

We believe that the overall update requirement in a constantly evolving hypermedia corpus is expected to be drastically reduced by utilizing ReferenceIDs pointing to other DLs (instead of those pointing to DSs) and relevant LocationTypes (instead of absolute LocationTypes). A somewhat similar approach has been followed in [13] with the introduction of location (LocSpec) and reference specifiers (RefSpec). The idea that influenced the definition of DL is not new. It is termed “transclusion” [24] and has been proposed from hypertext visionaries but, thereafter, ignored by hypermedia implementers. To avoid confusion with composite nodes, we stress that, although composites intrinsically refer to whole Nodes, transclusions refer to parts of a Node. A good example of the differences between the two notions, is given in [25]: authors could build some policy documents using composites; storing each clause as an atomic object and creating standard collections of clauses (composites) would allow authors to easily compose modular documents. But in the case of two documents differing only in a part smaller than a clause, authors would be forced to create two distinct clauses, identical in everything but the part, thereby loosing the notion (and operational benefit) that these clauses essentially are identical. Through the use of DLs the original clause needs not be copied and modified, as long as the appropriate DLs have been defined.

3.3 Anchors:

In the proposed model an Anchor instance is an object that participates in linking and is defined in the context of a specific Node. An Anchor could be thought of as the rendered part of a link, which can be activated in order to traverse the link. The properties of the class are:

- UID: Unique object identifier across hypermedia (masked).
- ServerID: The unique identifier of the server that has access to the Anchor instance (w-o).
- LocalID: The local identifier of the Anchor in the context of its parent Node (w-o, system provided).
- ParentID: The unique identifier of the Node that hosts the Anchor (w-o).
- DLocationID: The unique identifier of the DL that provides the location for the Anchor. If left empty, then an anchor to the whole Node is defined.

Unlike other modeling approaches that bind an Anchor definition to a specific media content or type (either directly like Hyper-G [8] or indirectly like the Dortmund Family [26]) we favor the de-coupling of the Anchor definition from any media-specific information, through the DLocationID property. This ensures that an Anchor remains a neutral, “hyper-“ and not a “-media” specific component of the overall model. The advantages can be better understood if we consider the next two cases:

1. A DL with a LocationType gets invalid due to changes of the DS content that it refers to. The Anchors that refer to this DL remain valid and the propagation of DS update is confined to the DL.
2. Even though the DS remains unchangeable, some properties of its LocationType have been updated (e.g., an image DL has been updated to represent some new area). All the same, the Anchor needs not to be updated.

3.4 Nodes:

A node instance is a wrapper-object, which maintains a set of references to other objects. The properties of the class are:

- UID: Unique object identifier across hypermedia (masked).
- ServerID: The unique identifier of the server that has access to the Node instance (w-o).
- LocalID: The local identifier of the Node in the scope of the server ServerID (w-o, system provided).
- DLocationID: A list of DL UIDs with binding information for ChildID elements. The media content of the Node comprises the resolved Content of the corresponding DLs and their child-Nodes. If the list is empty, then the ChildID must contain at least one element.
- ChildID: A list of Node UIDs. A Node may behave like a composite node and, in this case, the UIDs of the referred Nodes are included in the ChildID list. If this list is empty then the DLocationID must have at least one element. When a new element is added to the ChildID, the DLocationID is extended with all or selected DLocationID items of the child Node and the AnchorID is extended with all or selected AnchorID items of the child Node.
- AnchorID: A list of Anchor UIDs. The referred Anchors must be defined over DLs that are part of the Node (e.g., their UIDs are included in DLocationID list) or over DLs that are part of the Child nodes (e.g., their UIDs can be found is some DLocationID of the child Nodes).

We emphasize that the Node class combines the notions of both atomic and composite components as they have been described in the Dexter Reference Model [5]. Also, it covers the important aspect of customization. Before going into details of customization, we discuss the various methods for defining composite Nodes, e.g., Nodes that hold references to other Nodes.

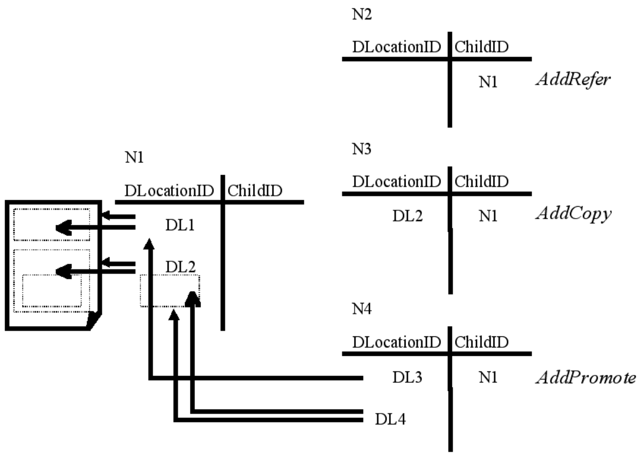

Figure 5: Three different types of composite Nodes created using AddRefer, AddCopy and AddPromote

The Dexter Reference Model has been criticized [27] for not making clear if a composite component stores a copy of the component it contains or if it refers to it. The proposed data model introduces the concept of DL, which deals with this issue in an elegant and explicit way. When adding a new Node

UID to the ChildID list of its parent Node, three distinct methods can be employed: AddRefer, AddCopy, and AddPromote. The AddRefer method simply updates ChildID list with the newly added Node's UID. The AddCopy method, apart from updating the ChildID list, also extends the DLocationID list by adding all (or selected) DL UIDs of the child Node. Finally, the AddPromote updates the ChildID list, then creates a new set of DLs that refer to all (or some) DLs of the child Node and, finally, appends them to DLocationID. The selection of the appropriate method substantially differentiates the behavior exhibited by the composite node throughout its lifetime. For example consider a child Node (N1) which comprises two DLs over a single DS (\h Figure 5). Child Node N1 has been added in its parent Node N2 by calling AddRefer. That implies that all the locations that have been included in N1 are also "visible" in N2 (by resolving, at run-time, the N1 entry of N2 ChildID list). In order to achieve greater flexibility, we add N1 in the ChildID of N3 using AddCopy. In this case, we have the option to exclude some portion of N1 that we do not want to be "visible" in N3 (i.e., DL1). Finally, N4 exhibits the use of AddPromote where the DL4 selects a sub-location of DL2, leading to a finer granularity of what the visible part of DL2 will be. Special provision must be taken for synchronizing the locations of a child node with the locations of its predecessors. In the case of AddRefer, no such need really exists, as the resolution ensures an always up-to-date view of a child. In the case of AddCopy and AddPromote, binding information is kept in the parent's DLocationID to associate a certain child UID with the relevant DLocationID elements. In the case that a location has been added (or removed) from a child Node, the parent Nodes capture the event (through a S2SOHP message) and cascade update in their DLocationID. Just like child Locations, all the Anchors of a child node may be "visible" to their predecessor or selectively some of them in case AddCopy or AddPromote were applied.

The proposed Node modeling capitalizes on the location concept and offers a great flexibility to construct simple atomic nodes or advanced composite nodes. In this last case, the user decides which Locations and Anchors will be "visible" in a composite node and, this, in turn, guarantees a better customization and adaptation to his information needs. On the other hand, most of the OHSs referred in this paper do not support composites and, in the case that composites are supported, they are restricted to refer to whole nodes (i.e., corresponding to the AddRefer method) thus, no substantial customization is supported.

3.5 Endpoints

An Endpoint is an object, which binds an Anchor to a Link object and stores traversing information, which can take one of the following values: source, destination or bi-directional. We favor this modeling approach, instead of expanding the Anchor class with directionality and link references, as it allows for Anchor re-use in more than one links possibly with different directionality (a similar approach has been adopted in [2], [14], [13] and [10]). An Endpoint instance has the following properties:

- UID: Unique object identifier across hypermedia (masked).
- AnchorUID: Anchor unique identifier. The relevant Anchor instance is part of the Link that the Endpoint specifies (w-o).
- LinkUID: Link unique identifier. The relevant Link instance includes the Anchor that the Endpoint specifies (w-o).
- Direction: Defines the directionality of the Anchor in the context of the specific link. It assumes one of the predefined values, source, destination and both. These values are interpreted on the client-side at run-time, and determine the navigational behavior of the client.

3.6 Links

A Link instance is a collection of other object references. The core of a Link is a list of references to the Endpoints that participate in the Link. The properties of the Link class are as follows:

- UID: Unique object identifier across hypermedia (masked).
- ServerID: The unique identifier of the server that has access to the Link instance (w-o).
- LocalID: The local identifier of the Link in the scope of the server ServerID (w-o, system provided).
- EndpointID: The list of the Endpoint UIDs that comprise the Link.
- DLocationID: A (possibly empty) list of DL UIDs, which provides meta-information about the link.

4. Using taxonomies and abstract models to evaluate Aristotle

Based on the OHS taxonomies and abstract hypermedia architectures, we may judge if a specific system is "open", i.e., if it conforms to the "openness" criteria that the taxonomy/ architecture poses. A well-known taxonomy is the Flag ([28], [29]) taxonomy for hypermedia systems. According to the Flag taxonomy a hypermedia system is "open" if it makes a clear decomposition of content-structure and storage-run time behavior. Using the Flag visual representation of a system, we may accurately distinguish between non-open systems (monolithic, like the HyperCard or with embedded links, like the WWW) and open systems (link service, like Microcosm, and Hyperbases, like Devise). An example of Flag-based system representations can be found in \h Figure 6.

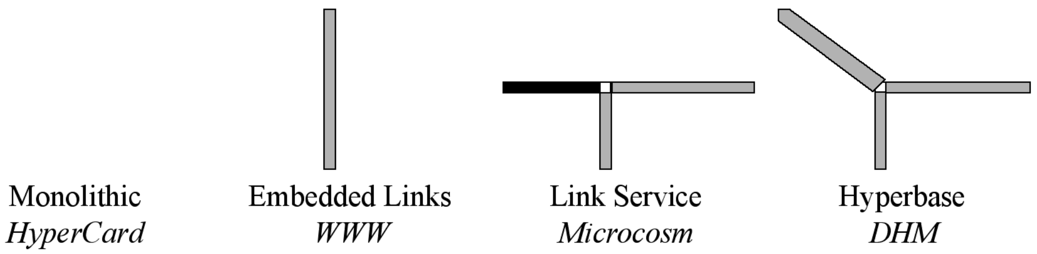

Figure 6: Examples of hypermedia system representations using the Flag taxonomy

According to Flag, the Aristotle architecture is a true OHS with Aristotle's entities mapping to the four corners of the Flag diagram (\h Figure 7). The open set of Aristotle clients maps to the "Viewer" corner, the CIM to the "Session Manager", the HSM to the "Data Model Manager" and, finally, the CAM to "Storage Manager". The gray-colored interfaces have also been formally defined and a one-to-one mapping exists. Specifically, the "Linking" interface corresponds to the "gateway"-to-client protocol, encoded in SISs. The C2SOHP maps to the "Presentation" interface and, finally, the "Storage" interface matches the HSM-CAM service-Connector set of communication specifications.

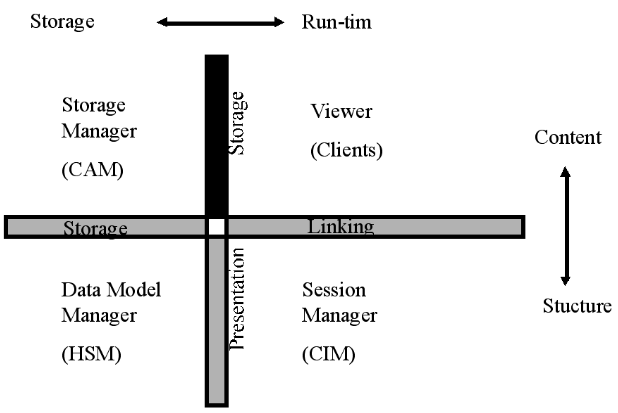

Figure 7: Aristotle’s representation according to the Flag taxonomy

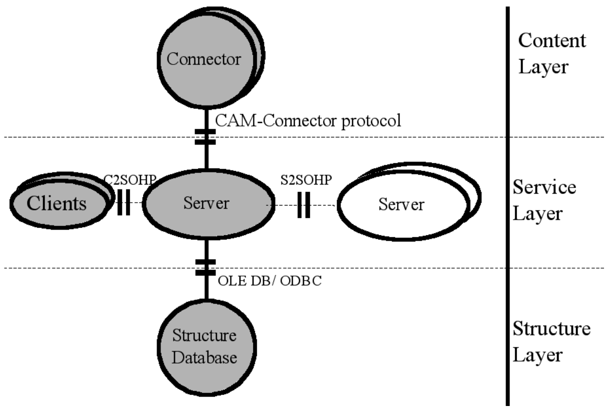

Figure 8: Aristotle’s representation according to CoReArc

The Common Reference Architecture (CoReArc) abstract architecture has been recently proposed [30]. CoReArc is influenced by the DHM/Dexter, HyperDisco ___ HOSS and extends Flag in the sense that new modules may be added and their interfaces be represented. Again, Aristotle is classified as open and may be mapped as \h Figure 8 shows. The crucial issue that CoReArc emphasizes, is to ensure multiplicity in architectural elements and to unambiguously define their interfaces. Aristotle achieves both objectives, by allowing multiple-entities to cooperate in “Content” and “Service” layer through well-defined, XML-based protocols.

A final remark has to be made, regarding the proposed Hypermedia Data Model and its expressiveness. The OHSWG has proposed a basic hypermedia model [14] for OHS. The Aristotle data model not only is expressively equivalent with OHSWG model but also, in certain aspects, exhibits greater flexibility and conceptual richness. To facilitate this comparison, a visual representation for the two models is provided in \h Figure 9.

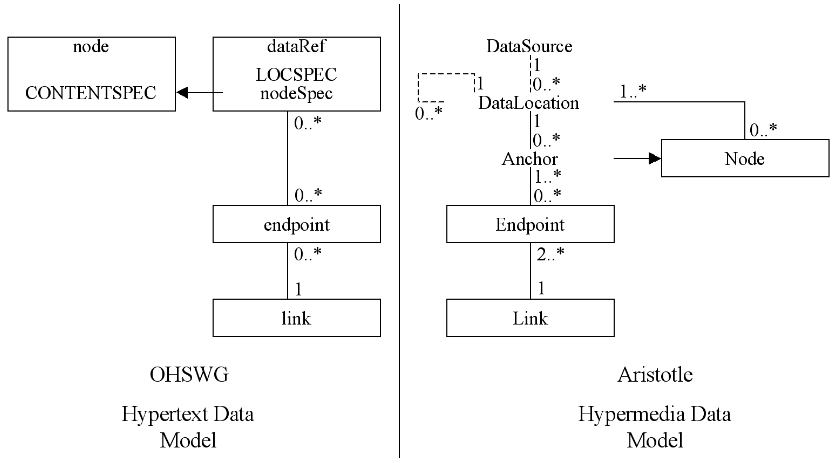

Figure 9: Diagrammatic representation of the OHSWG Hypertext Data Model and the proposed Hypermedia Data Model (Aristotle). The dashed line denotes a conditional existence of the relationship that is a DataLocation instance may refer to a DataSource or another DataLocation but not both.

The main difference of the two approaches is the modeling of data sources and data locations (termed CONTENTSPEC and LOCSPEC in OHSWG terminology). The OHSWG takes a quite different approach, than Aristotle, and models CONTENTSPEC as part of a Node definition. This results to the confusion of hyper- and media-specific notions and is accompanied by redundancy: for every node that refers to the same data source all information required to locate and access multimedia content needs to be re-entered. Apart from this redundancy effect and its consequences (e.g., need for multiple updates), this modeling implies only atomic nodes, which is a very restrictive assumption to face real-world problems.

The concept of LOCSPEC lies closely to that of the anchor and, in principle, aims to provide a media-depended addressing mechanism for rendering the anchor. We believe that this approach it is too restrictive for three reasons. Firstly, the same LOCSPEC has to be defined more than once if many anchors have been defined over the same media-location. Secondly, no composites are supported – the competitive advantage of Aristotle may be appreciated considering the various ways to define a composite node (addRefer, addCopy and addPromote methods). Finally, transclusion and location composition urges for the disassociation of the data location from any other object, i.e., the DLT should not be “embedded” in another object type or be part of it.

5. Conclusions and future work

Before concluding, it is important to examine how Aristotle fulfills the posed requirements for OHS. The scalability requirement is fully supported as storage limitations are treated externaly to Aristotle. The provision for an open set of data formats is also a feature of the underlying model and, as it has already been discussed, it is transparently ensured. The set of external (client) applications that may participate in the architecture varies from special Aristotle clients to third-party applications, but it is clear that in the latter case a reduction of the hypermedia functionality is the possible shortcoming.

Regarding the openness of the underlying data model, we remark that even though the Hypermedia Data Model is fixed, the Content classes are boundlessly extensible.

It is difficult to estimate if it is feasible to build an open OHS in terms of the underlying data model. This further complicates, as the trade-off of building “smart”/ “thick” clients must be also considered. Wide scale distribution, which is a major objective of the proposed architecture, is handled through a specially developed notification scheme that the Router module implements. Aristotle servers support multiple concurrent users and guarantee database locking for the Hyperbase. When the actual data are kept outside server’s jurisdiction, e.g., an HTML page that is under the control of an HTTPd, inconsistencies may occur. In such a case, the Server employs a detection policy, based on timestamps and file signatures. It is planned to investigate how heuristic techniques [31] might be employed to improve this policy with correction capabilities. All changes that affect a hypergraph are handled through C2SOHP notification messages, hence long-term inconsistencies are avoided.

The distribution of multimedia content over the Internet, the integration of various and heterogeneous data sources and the integration of third-party applications in a homogeneous and typical style are challenges where OHS have an unquestionable advantage over WWW. On the other hand, the WWW has matured significantly and the development of web content has become extremely easy as widely accepted standards and commercial software shores up for it. This paper presented an OHS architecture, which fits nicely into the posed requirements for OHS and retains compatibility with widely used and upcoming standards like Internet communication protocols and XML. This architecture, empowered by a novel hypermedia data model, also attempts to bridge the gap between more theoretic approaches and real-world implementations, which is considered to become an essential proving factor for OHS.

References

[1] W3C, HTML 4.0 Specification, W3C Recommendation (REC-html40-19980424). http://www.w3.org/TR/REC-html40/, Apr. 1998.

[2] K. Gronbaek and P. Orbaek. Webvise: Browser and Proxy Support for Open Hypermedia Structuring Mechanisms on the WWW. In WWW8 Conference, Toronto, Canada, 1999.

[3] T. Knopik and Bapat, A. The Role of Node and Link Types in Open Hypermedia Systems. In ECHT’94 Workshop on OHS, Edinburgh, UK, pages 33-36, 1994.

[4] A. M. Carvalho Moura et al. A survey on metadata for describing and retrieving Internet resources. Word Wide Web, Baltzer Science Publishers BV, 1:221-240, 1998.

[5] F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model, CACM, 37(2), 1994.

[6] H. Davis, A. Lewis and A. Rizk. OHP: A Draft Proposal for a Standard Open Hypermedia Protocol. In the 2nd Workshop on Open Hypermedia Systems (ACM Hypertext ’96), Washington D.C., 1996.

[7] H. Davis et al. Microcosm: An Open Hypermedia Environment for Information Integration. Technical Report CSTR 92-15, Dept. of Electronics and Computer Science, University of Southampton, England, 1992.

[8] H. Maurer. Hyperwave: The Next Generation Web Solution. Addison-Wesley, 1996.

[9] G. Hill, R. Wilkins and W. Hall. Open and Reconfigurable Hypermedia Systems: A Filter-based Approach. Technical Report CSTR 92-12, University of Southampton, 1992.

[10] U. Wiil and J. Leggett. The HyperDisco Approach to Open Hypermedia Systems. In ACM

Hypertext ’96, Washington D.C., 1996.

[11] U. Wiil and J. Leggett. Workspaces: The HyperDisco Approach to Internet Distribution. In ACM Hypertext ’97, Southampton, UK, 1997.

[12] K. Anderson, R. Taylor and J. Whitehead. Chimera: hypertext for heterogeneous software environments, In ACM ECHT’94, 1994.

[13] K. Gronbaek and R. Trigg. Toward a Dexter-based Model for Open Hypermedia: Unifying Embedded References and Link Objects. In ACM Hypertext ’96, Washington D.C., 1996.

[14] Open Hypermedia Systems Working Group. Navigation Domain: A Common Data Model. http://www.daimi.aau.dk/~pnuern/ohs/domains/navigation/model.html, 1998.

[15] S. Goose et al. An Open Framework for Integrating Widely Distributed Hypermedia Resources. In IEEE Multimedia Computing and Systems Conference, Hiroshima, Japan, 1996.

[16] S. Goose et al. Microcosm TNG: A Distributed Architecture to Support Reflexive Hypermedia Applications. In ACM Hypertext’97, Southampton, UK, 1997.

[17] L. Carr, W. Hall and S. Hitchcock. Link Services or Link Agents?. In ACM Hypertext’98, Pittsburgh, 1998.

[18] L. Carr et al. Implementing an Open Link Service for the World-Wide Web. World Wide Web, 1(2):61-71, 1998.

[19] W3C. Extensible Markup Language (XML) 1.0 - W3C Recommendation. http://www.w3.org/TR/1998/REC-xml-19980210, 1998.

[20] F. Kappe. A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems. Journal of Universal Computer Science, 1(2):84-104, 1995.

[21] C. Petrou, D. Martakos and S. Hadjiefthymiades. Adding Semantics to Hypermedia towards Link’s Enhancement and Dynamic Linking. In Hypertext-Information Retrieval-Multimedia ’97 (HIM’97) Conference, Dortmund, pages 175-190, 1997.

[22] C. Petrou, D. Martakos and Hadjiefthymiades, S. Hypermedia Linking and Querying: A Fuzzy-based Approach. In IEEE International Conference on Systems, Man and Cybernetics (SMC’98), San Diego, California, pages 1235-1240, 1998.

[23] Knowledge Based Systems, Inc. IDEF1x Design Method Report. 1995.

[24] T. Nelson. The heart of Connection: Hypermedia Unified by Transclusion, CACM 38(8): 31-33, 1995.

[25] M. Bieber et al. Fourth Generation Hypermedia: Some missing Links for the World Wide Web. Int. Journal on Human-Computer Studies, 47:,31-65, 1997.

[26] K. Tochtermann and G. Dittrich. The Dortmund Family of Hypermedia Models – Concepts and their Application. Journal of Universal Computer Science (J.UCS), 2(1), 1996.

[27] J. Leggett and J. Schnase. Viewing Dexter with Open Eyes. CACM, 37(2), 1994.

[28] U. Wiil and K. Osterbye. In ACM ECHT’94 Workshop on Open Hypermedia Systems, Technical Report R-94-2038, 1994.

[29] K. Osterbye and U. Wiil. The Flag Taxonomy of Open Hypermedia Systems. In ACM Hypertext ’96, Washington D.C., 1996.

[30] K. Gronbaek and U. Wiil. Towards a Common Reference Architecture for Open Hypermedia. In Journal of Digital Information (JoDI), 1(2), 1998.

[31] J. Vanzyl et al. Open Hypertext Systems: an Examination of Requirements, and Analysis of Implementation Strategies, Comparing Microcosm, HyperTED and the WWW. http://www.inf-wiss.uni-konstanz.de/Res/openhypermedia.html, 1994.

[32] P. Nurnberg and J. Leggett. A Vision for Open Hypermedia Systems. Journal of Digital Information (JoDI), 1(2), 1998.

[33] K. Gronbaek and U. Wiil. Towards a Reference Architecture for Open Hypermedia. In the 3^{d} Workshop on Open Hypermedia Systems (ACM Hypertext ’97), Southampton, UK, 1997.