Auld Leaky: A Contextual Open Hypermedia Link Server

Danius T. Michaelides, David E. Millard, Mark J. Weal, David De Roure

Intelligence, Agents, Multimedia,
Dept. of Electronics and Computer Science, University of Southampton, U. K.
{dtm, dem, mjw, dder}@ecs.soton.ac.uk

Abstract. The work of the Open Hypermedia Systems Working Group (OHSWG) has lead to the creation of several hypermedia models and a common protocol for Navigational Hypertext. However none of these include a working model of context. In this paper we present how we have extended the Fundamental Open Hypermedia Model (FOHM) to include context and behaviour. We then present Auld Leaky, a lightweight contextual link server that stores and serves structures represented in FOHM, using Context to filter query results.

1 INTRODUCTION

The Open Hypermedia Systems Working Group (OHSWG) has spent several years developing and defining the Open Hypermedia Protocol (OHP) [3] in an attempt to achieve interoperability between Open Hypermedia Systems. This development effort relied on both the definition of a suitable Hypermedia model and also a protocol (either text or API-based) to manipulate it. As Hypermedia is a wide field, the protocol was split into several different domains (Navigational, Spacial and Taxonomic) and OHP-Nav was developed as a text-based protocol that addressed only the Navigational domain. However, it has been argued that the community should have concentrated on the model rather than the protocol [11], particularly as several areas of the model, including Context, Behaviour and Computation, were never formally agreed.

Individual research groups within the OHSWG have taken the OHP-Nav model and extended it. Within the IAM group at Southampton we have developed the Fundamental Open Hypermedia Model (FOHM) [12] based on the OHP model, but capable of representing structures from all the discussed domains.

FOHM was always intended to include Context and Behaviour [10]. Recently we have turned our attention to producing a firm definition of context in FOHM and producing a contextual link server that would be flexible and general enough to use within several projects within our lab.

Version one of this link server will be known as Auld Linky, however the beta-version we are currently using is affectionately known as Auld Leaky.

1.1 Requirements

Before construction of Auld Leaky began we set down several simple requirements. These were based around our main goal of creating a simple link server that could be easily installed. This meant that we would be aiming at a single executable with a simple API. We wanted the link server to be as lightweight as possible and not require large infrastructures such as databases or middleware systems. The link server should just serve links with no need to access external services. Finally, and most importantly, the link server needed to act contextually. To this end we envisaged extending FOHM to allow context and behaviour objects to be attached to each of the hypermedia objects.

2 FOHM

In its work on interoperability, the OHSWG considered the requirements of several domains of hypertext. The three most frequently mentioned were Navigational [6, 1, 7, 15, 2], Spatial [9, 8, 14] and Taxonomic [13, 4, 5]. The OHP protocol was always more concerned with Navigational Hypertext, however FOHM, which is based on the OHP-Nav data model, is capable of expressing all three domains.

2.1 The Model

In FOHM we describe four first-class objects that are analogous to the objects in the OHP data model. *Associations* are structures that represent relationships between *Data* objects. These Data objects are wrappers for any piece of data that lies outside of the scope of the model. They normally represent a document although one could represent any file, stream or other item.

Data objects are not directly placed in the Associations. Instead *Reference* objects are used, these either point at Data objects in their entirety or at parts of those Data objects, for example the second paragraph of a text document, or the second scene of a film. They are attached to the Association object via *Bindings*. Each Association also has a structure type and a feature space; each Binding must state its position in that feature space, effectively stating how it is bound to the Association's structure. A FOHM Navigational structure is illustrated in Figure 1.

Essentially, an Association relates different objects. The Bindings attach References to the Association. The References point to objects in the system. In the illustration the objects are items of Data, but References can also point to other Associations.

2.2 Navigational Hypertext in FOHM

Navigational Hypertext is the most common of the domains that can be represented in FOHM. Its notion of directed links can be modelled easily by a single Association feature "direction" to which Data objects are bound with either a "source", "destination" or "bi-directional" value.

Figure 1. shows a Navigational structure described within the FOHM model, in this case two links. The first is a link across three different data objects (one of which is referenced in its entirety), the second is a link across one area of one document and three different areas within a second document. Notice that Associations can share References and that References can share Data objects.

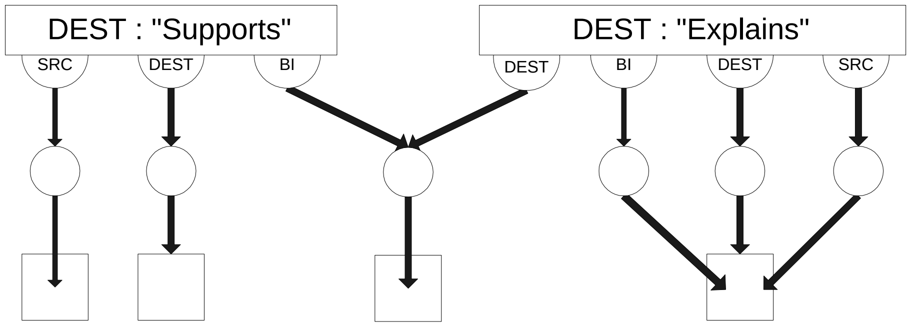


Fig. 1. Link Structures in FOHM

3 AULD LEAKY

The Auld Leaky implementation is written in Perl, it consists of a number of components that can be compiled into a single executable. The core is built around a FOHM implementation which provides basic container objects and APIs for matching structures together. FOHM objects are grouped into linkbases, which are managed by the link server. The link server provides APIs for storing, looking up and querying stored linkbases. The final component consists of a module to expose the link server as a web-server-like process, using HTTP and XML.

3.1 Communication

Auld Leaky communicates using FOHM structures that have been serialised into XML. These are then sent over HTTP connections. Similar to a web server, Leaky responds to the standard HTTP request types:

`GET`	requests are sent with an ID (a simple string), the relevant object is located in the linkbase and returned in the form of XML.
`POST`	requests are sent with a FOHM object in the content of the message. This is then pattern-matched against the objects in the linkbase (this is explained further in Section 3.2).
`PUT`	requests are sent with a FOHM object in the content of the message. Leaky then adds this object to the linkbase.
`DELETE`	requests are sent with an ID, the relevant object is located in the linkbase and removed.

3.2 Pattern Matching

The basic mechanism for querying in Auld Leaky is pattern matching. To perform a query, a FOHM structure is constructed and then matched against each structure stored in the link server. The matching of two structures proceeds in a tree-like prefix depth first search, meaning that when matching two FOHM objects, first each of their attributes are matched, followed by the matching of any sub-objects. Matching immediately fails when any attribute or object matching fails.

In order to facilitate powerful queries, we add extra attributes to the FOHM objects. Firstly, an attribute called `state` indicates the nature of the structure with respect to matching. The attribute has four values; *undefined* and *defined* indicate whether the particular object is in a defined state or not (where being undefined means being set to nothing, or null). A third state, *variable*, marks an object as being a variable. Variables are objects that match any other object of the same type, they are currently anonymous, and can contain some form of constraint to allow for complex matching. Our current implementation allows for constraints expressed as perl expressions (utilising perl's `eval` in a sandbox), but other constraint languages and suitable evaluators could be used. The fourth value of state, *lookup*, is used for linkbase loading, and not for matching purposes. Linkbase loading is discussed in section 3.5.

Whilst the use of the state attribute is sufficient for constructing queries, it can often lead to quite verbose structures. Specifically, queries often contain just a few concrete values and the remaining values are variable or just not set to anything (for example an author might want to create an Association without a description attribute). To avoid this, we introduce an extra attribute, called `missing` which has two values: *variable* and *undefined*. This attribute indicates what state a field in an object should assume if the field is unspecified.

Note that our query objects are FOHM structures which means that we can store them in linkbases.

3.3 Matching Bindings

The Bindings contained within an Association are a special case for matching purposes. The Bindings in an Association are a bag of Binding objects. This presents a problem to our matching algorithm, since there is a choice. For all other FOHM structures there is a strict correspondence between objects in each structure to be matched. To match Binding objects, we have to employ an exhaustive search algorithm to try and find a solution such that all Bindings have been matched with a Binding in the other Association. The algorithm uses a recursive method of attempting to match a Binding to each available Binding in the other Association. For each successful match, the Bindings are marked as unavailable and a recursive call is made. If the recursive call returns unsuccessfully, then matching continues with the next available Binding; if there are none, then the routine fails. If the recursive call returns successfully then a configuration of Binding matches has been found where all the Bindings have been matched.

Binding objects also have extra attributes to control how they are matched. The `optional` attribute indicates that a Binding does not have to be matched against another Binding. For the matching algorithm, this attribute simply means that it can ignore optional bindings and return successfully if only optional Bindings remain unmatched.

The `repeatable` attribute allows for a Binding to be matched against more than once. This is typically used to match against Associations where the number of Bindings in unknown. For example, the right hand Association in Figure 1. has a `src`, two `dest` and a `bi` (directional) Binding. A query to find all the links from the source document would include the `src` Binding but it would also include a variable Binding for the `dest` Bindings to match against. In this case the variable Binding is marked as repeatable, so that it can match any number of `dest` Bindings.

3.4 Querying

The process of performing a query entails constructing a FOHM structure to be used in the pattern matching and making a `POST` HTTP call to the link server. The link server receives the XML data and converts it into an internal FOHM structure. This is then matched against each object in the linkbase; any matches are accumulated. It is possible for the query to specify the linkbase or scope the search by specifying a more restrictive path in the HTTP request. The matching results are converted to XML and returned to the client. The client will then typically reconstruct the results back into FOHM objects.

3.5 Linkbases, Naming and Loading

Associations, References and Data objects are first class in FOHM; they have an `id` value. Linkbases are collections of FOHM structures. We try and preserve URL semantics so that an object can be referenced by:

```
http://servername:port/<linkbase name>/<object id>
```

The naming scheme used within the linkbases is up to the author, but using a hierarchy, if appropriate, is consistent with the URL scheme. Structures can also have local identifiers denoted by `#value`. Local identifiers are given to structures which do not have their own identity, but need to be referenced, e.g. for cyclic structures. The identifier is only valid within the scope of the communication.

Links are typically loaded into Auld Leaky when the service is started, but can also be stored using the `PUT` method. Note that since link servers load up their structures from URLs, its possible that a URL can be supplied that points to another link server, possibly even a query.

Structures stored can be references to other objects, using the `lookup` value of the state attribute. The `id` attribute on such structures, indicates the destination of the reference. Primarily, lookup structures are used for shared objects or for separating out the FOHM structures for easier authoring of linkbases. An open issue is the action of the linkbase when faced with a unknown identifier. The `id` could be a URL; this leads to a number of distributed link service possibilities.

The simplest case is that the link server would just store that it is a reference to an unknown object (although the type is known). The reference would have to be resolved by the client. During the matching process its not clear how the link server should treat the reference. The link server is faced with the problem of matching a (known) FOHM object against just a reference.

It could just assume that the structure matches and let the client decide for itself. An alternative would be for the link server to make a query to the remote link server passing on the structure to be matched; a kind of distributed search/query. This sounds quite attractive, but it does required that the reference points to a link server capable of performing the query. This approach is the subject of future work.

An alternative solution to the problem of a reference is to lookup the object from the specified location and store that structure locally. Hence when a structure is loaded into the link server, the server will also load remote structures referenced, so it has all the data it requires. This process could possibly be recursive. The link server effectively has a copy of the actual remote structures, and this raises a number of caching and consistency issues. This solution does mean that the link server is able to perform its matching unhindered.

3.6 Context and Behaviour

Context and Behaviour objects can be attached to all parts of the FOHM structure as is shown in Figure 2. These objects are opaque to the FOHM description and are defined separately. Any link server that deals in FOHM structures is required to understand the Context definitions, however the Behaviour objects are used only by clients and need not be understood (although they still have to be served).

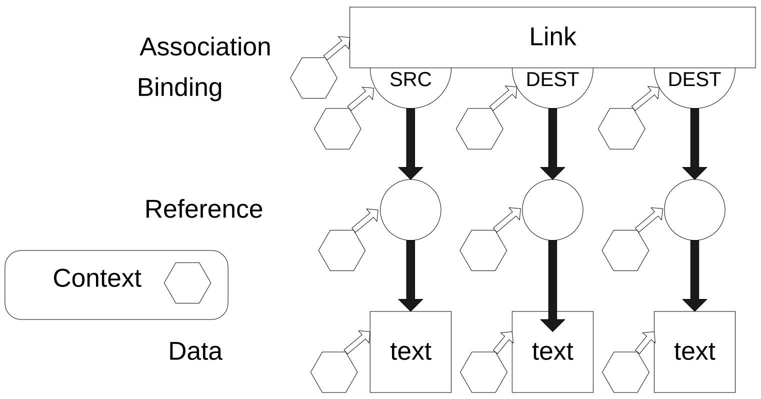


Fig. 2. Context Placement on FOHM Data Structures

Auld Leaky implements the context object of the FOHM structures as a set of key value pairs called contextvalues. Although a relatively simple mechanism for representing context it is deceptively powerful. Figure 3. shows an example context object.

From this Context object the following information can be identified:

1. The value of context item location is given as “office”.
2. The context item time has a constraint attached to it that defines time as being later than 19:05.

This last time-based context is an interesting example. Notice that the type of the object is not specified, i.e. the time is represented as a string 19:05. The contextvalue

```
<context>
  <contextvalue key="location">office</contextvalue>
  <contextvalue key="time" state="variable">
    <constraint> ( $_[0] gt "19:05"); </constraint>
  </contextvalue>
  <constraint>(!$_[0]->contextvalue("passed_deadline"));</constraint>
</context>
```

Fig. 3. XML Representation of Context Object

```
<behaviour event="display">
  <behaviourvalue key="passed_deadline">true</behaviourvalue>
</behaviour>

<behaviour event="traversal">
  <behaviourvalue key="location">home</behaviourvalue>
  <behaviourvalue key="time">+00:05</behaviourvalue>
</behaviour>
```

Fig. 4. XML Representation of Behaviour Object

also has a constraint attached to it. Constraints can be used on any element where the default matching (a string comparison) is not sufficient, in this case by invoking a greater than comparison.

The Context object itself also has a Perl constraint that indicates that a successful match requires that the contextvalue with the key "passed_deadline" should not exist.

Behaviour objects are implemented in Auld Leaky in much the same way as Context objects.

For example, Figure 4. shows two Behaviour objects. The first could be attached to a Data object. Each Behaviour object has an event type associated with it. In this case, the event "display" is used to indicate that the Behaviour should be carried out when the object is displayed. This is client dependant and in no way enforced by Auld Leaky. In this case the client interprets the Behaviour in such a way that upon display of this Data item it sets the "passed_deadline" flag in the clients context to be true.

The second Behaviour object could be attached to an Association object. The event here is traversal rather than display. When the client follows the Association above the value of time in the client's context will be increased by five minutes and the location is changed to home.

These two Behaviour objects are fairly simple and both result in modifications to the client's user context which are then used to carry out further matching in the application in which they are used. The Behaviour object need not be used in this way and could just as easily be constructed to trigger additional events in the system or to allow objects to express the specifics of how they are displayed.

3.7 Querying in Context

There are two important ways that Context can be used in Auld Leaky. Firstly, since Context objects can be attached to FOHM structures it becomes possible to attach them to the structures sent in queries, these structures then only match if their contexts match as well. Secondly, Context objects can be attached to the queries themselves. This Context object acts like a filter on query results. Objects that are returned from the query matching are matched against the supplied query Context. Objects whose Context does not match the query Context are removed from the result set. The Context matching filters down the FOHM structures so that sub-objects can be removed.

For example, this means that if a Binding matches the Context but that the Data object it binds does not, then the Binding, the Data object and the Reference that joins them are all removed from the results set.

An issue raised with this context filtering is that a structure could be filtered such that it would no longer match the original query. For example, a query that required a source binding to a particular URL could find that, due to filtering, all the source bindings had been removed.

The solution we have implemented to this problem is to re-match any structures that get filtered against the original query.

4 APPLICATIONS

As well as allowing contextual filtering of Navigational links it is possible to use Auld Leaky as a contextual structural engine to drive novel applications. In this section we shall briefly explain some of things we have been exploring.

4.1 Hypertext Short Stories

One of the applications constructed using Auld Leaky is the cgi based Hypertext Story Engine. This constructs short stories for the reader on the fly from data and links stored within the linkbase. The client maintains a user profile, in the form of a context, that it uses to find the next appropriate story fragment.

A story is constructed from fragments of text. These are stored as data objects in the linkbase. To link all of these story fragments together, Associations exist within the linkbase. These Associations contain one source Binding that is totally generic (will match any Data item). However, the Association has a Context attached to it. The client constructing the story queries the linkbase with the user context and is returned all of the Associations that match that Context. Figure 5. shows the association structure used in the application.

The story fragments are not directly linked to each other. Instead, the client uses the current readers context to offer the reader a selection of story fragments to view. Viewing the story fragment will modify the readers context , based on Behaviour objects

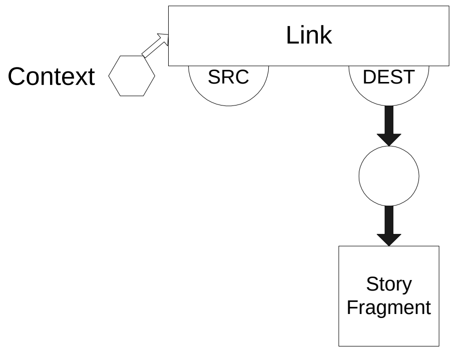


Fig. 5. The Short Story Link

attached to the Association and Data objects, resulting in a different set of fragments being offered next time.

We are currently using Context and Behaviour as a way of keeping track of time, characters and also plot progression (for example to restrict basic exposition and background information to the beginning of the story when the scene is being set).

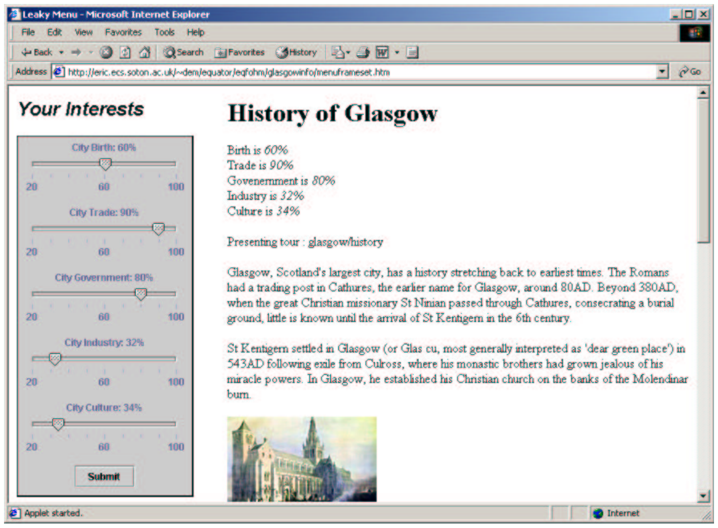
4.2 Contextual Information Systems

Auld Leaky can also be used as a way of accessing factual information. We designed an applet that dynamically builds a document about the history of the City of Glasgow. It does this based on the interest preferences that the user has selected (choosing from areas such as Culture, Industry and Government). Alongside the web server that serves the applet we run a copy of Auld Leaky which is responsible for providing the necessary FOHM structures to the applet.

The dynamic document is represented in FOHM as a transclusive tour (this means that the tour is compiled into a single document). Each item in the tour is a secondary Association which we call 'level of detail' (LoD). Each LoD is a list which contained Data items ordered by their length. Context objects attached to each Data item described how relevant that item is to each interest.

The applet uses the interests selected by the user to construct a user context and then queries the whole tour. The link server returns the tour with each LoD included but with Data items that do not match the context removed. The applet then uses the first positioned Data item in each LoD to build the dynamic document.

Figure 6. shows a screen shot of the applet. The user has set the sliders on the left to represent their interests and the applet has dynamically built a document describing the History of Glasgow on the right, using the tour and LoD structures served up by Leaky.


Leaky Menu - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://eric.ecs.soton.ac.uk/~dem/equator/eqfohm/glasgowinfo/menuframeset.htm
Go
Your Interests
City Birth: 60%
City Trade: 90%
City Government: 80%
City Industry: 32%
City Culture: 34%
20 60 100
Submit
Applet started.
Internet

History of Glasgow

Birth is *60%*
Trade is *90%*
Govenernment is *80%*
Industry is *32%*
Culture is *34%*

Presenting tour : glasgow/history

Glasgow, Scotland's largest city, has a history stretching back to earliest times. The Romans had a trading post in Cathures, the earlier name for Glasgow, around 80AD. Beyond 380AD, when the great Christian missionary St Ninian passed through Cathures, consecrating a burial ground, little is known until the arrival of St Kentigern in the 6th century.

St Kentigern settled in Glasgow (or Glas cu, most generally interpreted as 'dear green place') in 543AD following exile from Culross, where his monastic brothers had grown jealous of his miracle powers. In Glasgow, he established his Christian church on the banks of the Molendinar burn.

Fig. 6. The Level of Detail Applet

5 CONCLUSION

In this paper we have presented Auld Leaky, a contextual Link Server that is architecturally light and yet has the ability to manipulate structures from multiple hypertext domains in a consistent way.

Although Leaky's use of context and behaviour is quite sophisticated the actual model of context used (a vector of key value pairs) is simple. We are currently working to produce a more complex ontology of context that would make some of the semantics and relationships within Context objects explicit.

The Architectural simplicity of Leaky also makes it an ideal component in a peer to peer distributed system. There are a number of opportunities for incorporating distributed aspects into the link service. In particular further work is needed to provide a strong naming strategy and investigate issues of caching structures. In addition we have identified a number of interesting distributed query issues which need to be explored.

We are also interested in developing the use of context. In particular in regard to narrative structures, where we are looking at using context to produce dynamically generated 'tours' of material where the tour contains a narrative structure with a recognisable 'beginning', 'middle' and 'end'. Such dynamic guides can also be tailored towards users' interests and may vary according to a required length (e.g. a user might request a five minute description). The Glasgow applet and Hypertext Story Engine represent our initial work in this area.

6 ACKNOWLEDGEMENTS

This research is funded in part by EPSRC IRC project ”EQUATOR” GR/N15986/01 and EPSRC DIM project ”LinkMe” GR/M25919.

References

1. Kenneth M. Anderson, Richard N. Taylor, and E. James Whitehead. 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*, pages 94–197, 1994.
2. Tim Berners-Lee, Robert Cailliau, A. Luotonen, Henrik Frystyk Nielsen, and A. Secret. The World Wide Web. *Communications of the ACM*, 37(8):76–82, 1994.
3. Hugh Davis, Siegfried Reich, and David Millard. A proposal for a common navigational hypertext protocol. Technical report, Dept. of Electronics and Computer Science, 1997. Presented at 3.5 Open Hypermedia System Working Group Meeting. Aarhus University, Denmark. September 8-11.
4. Parunak H. Van Dyke. Don't Link Me In: Set-Based Hypermedia for Taxonomic Reasoning. In *Proceedings of the '91 ACM Conference on Hypertext, Dec. 15-18, 1991, San Antonio, TX*, pages 233–242, 1991.
5. Parunak H. Van Dyke. Hypercubes Grow on Trees (and Other Observations from the Land of Hypersets). In *Proceedings of the '93 ACM Conference on Hypertext, Nov. 14-18, 1993, Seattle, WA*, pages 73–81, 1993.
6. Andrew M. Fountain, Wendy Hall, Ian Heath, and Hugh C. Davis. MICROCOSM: An Open Model for Hypermedia With Dynamic Linking. In A. Rizk, N. Streitz, and J. André, editors, *Hypertext: Concepts, Systems and Applications (Proceedings of ECHT'90)*, pages 298–311. Cambridge University Press, 1990.
7. Kaj Grønbæk and Randall H. Trigg. Design issues for a Dexter-based hypermedia system. *Communications of the ACM*, 37(3):40–49, February 1994.
8. Catherine C. Marshall and Frank M. Shipman. Spatial Hypertext: Designing for Change. *Communications of the ACM*, 38:88–97, 1995.
9. Catherine C. Marshall and Frank M. Shipman. Spatial Hypertext and the Practice of Information Triage. In *Proceedings of the '97 ACM Conference on Hypertext, April 6-11, 1997, Southampton, UK*, pages 124–133, 1997.
10. David Millard and Hugh Davis. Navigating Spaces: The Semantics of Cross Domain Interoperability. In Siegfried Reich and Kenneth M. Anderson, editors, *OHS 6 and SC2, Proceedings of the ..., Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743)*, pages 129–139, 2000.
11. David Millard, Hugh Davis, and Luc Moreau. Standardizing Hypertext: Where Next for OHP? In Siegfried Reich and Kenneth M. Anderson, editors, *OHS 6 and SC2, Proceedings of the ..., Published in Lecture Notes in Computer Science, (LNCS 1903), Springer Verlag, Heidelberg (ISSN 0302-9743)*, pages 3–12, 2000.
12. David E. Millard, Luc Moreau, Hugh C. Davis, and Siegfried Reich. FOHM: A Fundamental Open Hypertext Model for Investigating Interoperability Between Hypertext Domains. In *Proceedings of the '00 ACM Conference on Hypertext, May 30 - June 3, San Antonio, TX*, pages 93–102, 2000.
13. Peter J. Nürnberg, Erich R. Schneider, and John J. Leggett. Designing digital libraries for the hyper-literate age. *Journal of Universal Computer Science*, 2(9), 1996.
14. Olav Reinert, Dirk Bucka-Lassen, Claus A. Pedersen, and Peter J. Nürnberg. 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*, pages 49–50, February 1999.
15. John L. Schnase, John L. Leggett, David L. Hicks, Peter J. Nuernberg, and J. Alfredo Sánchez. Design and implementation of the HB1 hyperbase management system. *Electronic Publishing—Origination Dissemination and Design*, 6(1):35–63, June 1993.