THE GAIA FRAMEWORK: VERSION SUPPORT IN WEB BASED OPEN HYPERMEDIA

Thomas Kejser *Department of Computer Science, University of Århus, Denmark*

Kaj Grønbæk *Department of Computer Science, University of Århus, Denmark*

ABSTRACT

The GAIA framework prototype, described herein, explores the possibilities and problems that arise when combining versioning and open hypermedia paradigms. It will be argued that it - by adding versioning as a separate service in the hypermedia architecture – is possible to build consistent versioning of both documents and hyperstructure. This approach eases some of the problems inherent in separating versioning of documents and structure - and even allows versioning to be added to domains that would otherwise be unversioned. A brief introduction is given to research results in the hypermedia versioning field and GAIA is compared with previous attempts at defining hypermedia versioning frameworks. GAIA is capable of multi-level versioning and versioning of structures and supports freezing mechanisms for both documents and hyperstructure. The experiences from GAIA provide an input to new reference architectures for future versioned hypermedia frameworks.

KEYWORDS

Open Hypermedia, Versioning Frameworks, Hypermedia Architecture.

1. INTRODUCTION

Document versioning is an old discipline [5] that has been employed extensively in collaborative software development projects (see [12] for a taxonomy and overview). When developing and deploying commercial software several problems of versioning and storage present themselves. Solutions have been successfully modeled in several frameworks [25] [11] [6] [23]. Even though versioning has been treated in several papers [16] [21] [24] [26], a framework for providing open hypermedia versioning has not yet evolved. Versioning has been viewed as a problem to be solved in the underlying document storage mechanism or as

an extension to a hyperstructure service. These strategies have weaknesses, since the connection between documents and structure is effective severed by letting both documents and structure live their timelines inside separate (often loosely coupled) storage areas. Furthermore, the approach gives rise to new versioning problems: transitive freezing and structural versioning. A more in depth description of these issues is found in section 2.

Since 1989 the open hypermedia community has been working on allowing readers to annotate and add links to documents stored in a wide variety of repositories. It has been realized that the special purpose hypermedia editors used in monolithic systems (such as Notecard [19]) are not sufficient for the needs of all users. By integrating third party applications with hypermedia databases - users can continue to work with their accustomed tools. This integration strategy has led to one of the main "open hypermedia" requirements: that links and anchor structures are stored in repositories outside the documents if a hypermedia system is to be considered "open". The Web, due to its underlying technologies, is a special challenge when pursuing this strategy of integrating link and annotation systems with external storage.

Open hypermedia researchers have produced a number of frameworks built to meet the requirements of third party storage integration to the Web. An overview of the frameworks that have attempted versioning is given in [27] and [24]. Of the hypermedia versioning frameworks developed (or sketched) so far only Microcosm and Chimera can be considered "open", in the sense that they are capable of integrating with versioned document storage mechanisms outside of the hypermedia framework itself. A treatment of the version strategy of these frameworks is described in Section 3. But the openness is "broken" when versioning is added to the document storage mechanism. The GAIA framework defines a new architecture for versioned hypermedia frameworks. Versioning is treated as an external service in GAIA. Like open hypermedia itself, versioning is considered a service that can be added to any information repository. GAIA's design and architecture is found in Section 4. Section 5 discusses related work and Section 6 concludes the paper.

2. VERSIONING IN HYPERMEDIA

The hypermedia versioning discipline can be considered a superset of the discipline of software configuration management (SCM). If hypermedia is to be the universal paradigm for managing huge information databases, it must at least be able to support the demands of SCM. We upon the rich experiences of the SCM-researchers and look for inspiration there when designing hypermedia versioning.

However, hypermedia versioning is more than SCM: where SCM is used mainly for source code and software development processes, hypermedia deals with potentially any produced media type and work area. This posits a demand for a flexible implementation of the versioning paradigm. Nevertheless, we can adapt some the concepts defined by SCM (taxonomy here is inspired by [12]):

- **Version** – The state of a given document at a point in time.
- **Revision** – a version that is meant to replace an older version
- **Mutability** – The ability to modify a document version. The process of making a version immutable is known as freezing.

- **Baseline** – A collection (sometimes computed from a query) of one or more versions. The collection itself can be frozen and versioned (in some frameworks).
- **Version graph** – A graph describing the historical or logical relationship between versions. When there is more than one successor of a version (e.g. if two developers are concurrently working on a version) this is known as a **branch**. When two or more versions are joined into a new version it is called a **merge**.
- **Multi level versioning** – The ability of a version framework to support versioning of versions.
- **Cross document versioning** – The ability of a versioning framework to support the merging of versions from two different version graphs (This term is the authors own, since the taxonomy in [12] does not provide a name for this concept.).

Introducing these versioning concepts to hypermedia implies new problems– these are described below.

2.1 Freezing

The ability to create baselines has been a prime requisite the SCM frameworks since their inception – hypermedia will of course need similar constructs. As an example of the problems caused by introducing baselines into hypermedia, consider the following situation:

James is a programmer working on a document (D) describing the current status of a software module in development. He is annotating it with links (L_1, L_2, …, L_n) pointing to documentation for modules (also in development) that his module depends on (A_1, A_2, …, A_n). At some point James decides that document D is ready for publishing and asks the versioning framework to freeze it. What should happen?

1 *Freeze D and the transitive closure of all links passing out from D* – While this truly freezes everything that D refers to (and thus guarantees that the correct version of the modules referred can be found in the future), it imposes a problem of performance and concurrency on the versioning framework. Firstly, the transitive closure might be very large depending on the size of the docuverse in which D participates (e.g. consider the consequence of freezing the transitive closure of all links emitting from Yahoo). Secondly, authors working on the documents being forcefully frozen (for example A_1, A_2, …, A_n) as a consequence of the D-freeze will have to create a new version of their documents. This will create a branch in the document history and require a merge of their newly added information, thus forcing the creation of new revisions of possibly many documents. The consequences of such a course of action may be greater than James had expected – especially so if the documents that are frozen are not owned by any of James co-workers. Furthermore, merging documents have a significant cognitive overhead [31].

2 *Freeze D, L_1, L_2, …, L_n and A_1, A_2, …, A_n* – while this poses many of the same problem as in 1 it restricts the amount of documents that are frozen. The dependent modules in A_1, A_2, …, A_n may require other dependencies that are not frozen by the James' actions. This may lead to a inconsistency in James documentation. He may have created links to documents and source code that is not frozen – and thus the documentation may indirectly refer to objects that change.

3   *Freeze D and L_1, L_2, …, L_n.* – This scenario is much less restrictive than 1 and 2, but contains many of the problems from case 2. If someone changes A_1, A_2, …, A_n the links could have some sort of selection rule attached that will try to locate a the "best" version of A_1, A_2, …, A_n that fits the needs of D (for example the documents with version numbers closest to the publishing time of D). This does not guarantee the consistency found in 1 and 2 but allows for much less cognitive overhead. In addition to the problem of resolution, the semantics of links and computed endpoints becomes a new and important factor in the version framework.

One solution would of course be to require that all documents, once checked into the repository, should be automatically frozen. This would eliminate the freeze problem for documents and reduce it to that of freezing links. Any change to a document would then require the creation of a new revision. This approach is similar to the one proposed by Ted Nelson in the Xanalogical Storage model [22].

Østerbye [31] has suggested that we should be able to add new annotations and links to frozen objects. If the framework allows this, it will further complicate the freezing problem. A decision must be made whether annotations added to frozen documents should themselves be frozen when created or left as mutable objects.

2.2 Link Versioning

In hypermedia frameworks - anchors can be either embedded into documents or stored outside of them. If they are embedded, versioning of the document implies versioning of the anchors [13]. This is the approach of HTML, Adobe Acrobat [1], Notecard [19] and Neptune [9].

However, this approach is hardly feasible for open hypermedia systems where links and anchors must be stored outside of the documents. If the open approach is used, the known problem of anchors pointing to non-existent documents presents a challenge for the hypermedia designer. The addition of versioning further complicates this: Links may now become broken as a result of documents changing their version – unless the hyperstructure and document stores implement version graphs that are closely synchronized. Furthermore, the addition of versioning to the document store raises the question: "What should happen to the anchors in a document when a document changes its version?" Three answers come to mind:

1   Version both anchors and documents (much easier if anchors are embedded in the documents).
2   Let anchors point to a fixed version of a document
3   Adopt conventions for resolving anchor endpoints. E.g point to the "latest" version of document.

If choice 1 is made, the hypermedia version model must be kept in sync with the document version model. Choices 2 and 3 provide simple answers to anchor versioning. However, when anchors (and thus the hyperstructure itself) are not versioned we can no longer properly answer queries of the form: "Show me the state of document D *and* its surrounding structure to the time T".

2.3 Version Models

When designing a hypermedia framework several versioning features may be integrated, including:

- **Complexity of version graphs** – Should we allow only version trees (with no merging) or complete version graphs where branches in the graph can be merged.
- **Multi level versioning** – Can versions be versioned themselves?
- **Cross document versioning** – Shall we allow the version graph of one object to be merged with the versions graph of another object?
- **Granularity of versions** – What constitutes a version? In a hypermedia paradigm, it is hardly feasible to allow only a single object to be versioned. The framework should support versioning of sub structures of the hypermedia network. A baseline in hypermedia is (as implied in 2.1) not just a collection of versions, but may be a computed set of versioned and non-versioned objects.
- **Storage mode** – Should we use a delta based algorithm for storing version (and perhaps, like it is the case in Palimpsest [16] store only changes). Or should we use a pure state based versioning paradigm, always saving the complete state of all versions.

A hypermedia framework must implement a large subset of these models if it is to provide flexible version support for structures and documents.

3. PREVIOUS MODELS AND FRAMEWORKS

In this section an overview of earlier work in hypermedia versioning is presented. The frameworks that are treated are the ones that can be considered "open" in the sense that link and hyperstructure is stored in databases external to the documents. The GAIA approach is based on the belief that an "open" hypermedia system must be able to integrate with any storage system. The architecture of the framework must take this into consideration and thus cannot rely on the document storage subsystem to provide versioning capabilities.

3.1 Microcosm

The Microcosm [10] framework implements a large array of hypermedia functionality. Its basic architecture consists of a Document Control System and Filter Manager. The Document Control System serves as an interface to the docuverse, it allows for storage and retrieval both inside Microcosm and for storage retrieval on the Web (through HTTP or FTP). Microcosm researchers are working on achieving scalability and distribution of the link service, both on platform and process level.

Using its link API, Microcosm has integrated advanced hypermedia functionality in several third party applications. Web pages can be augmented using a special proxy service, which modifies pages before they reach the browser [20] [8]. Versioning support is limited, but work has been done into adding versioned links and documents using RCS [21].

Microcosm approaches versioning by relying on external versioning mechanisms – RCS for document versioning and its own mechanism for hyperstructure versioning. The versioning implemented requires all documents saved in the Document Control System to be frozen at save time. Links can be versioned and the user can choose what version of a link to follow, but links can point only to a specific version of a document. Melly and Hall [21] does not mention of how Web integration is to be achieved if RCS is not used as document and link repository.

The Microcosm versioning system is thus dictated by the capabilities of the RCS document subsystem. Thus Microcosm does not meet the demands of openness described in 1.

3.2 Chimera

Chimera has been described extensively in [2]. It provides open link services to a variety of applications and it has been tested in relatively large environments [4] where it faced some of the industrial requirements for managing hypermedia databases.

While Chimera does not directly support versioning, a proposal [26] on how to expand the API with the necessary functionality has been published. Versioning is to be achieved by adding "configurations" as first class objects of the hyperstructure. Configurations are collections of hypermedia objects and pointers to documents. Chimera leaves versioning control of the documents to external storage mechanism, and maintains a "version association table" to internally represent the docuverse in the hyperstructure

All hypermedia objects and document references can be a member of one or more configurations. When a hypermedia object is modified, new versions are created of all the configurations in which it is contained. Baselines are created by naming and freezing the current state of a configuration. Check-in and check-out can be done on all hypermedia objects, both links and anchors, but also configurations. This allows the hypermedia structure to be edited independently of the docuverse, but this is not without problems:

> *"The hypertext concepts must be capable of manipulation independent of the external system, and vice-versa. Hypertext versions will get out of synchronization with object versions -- a hypertext versioning system must accommodate this."*
>
> [26]

The version association tables in principle maintain some of this synchronization, but exactly how these tables are to be kept in sync with the underlying document storage is not discussed by Whitehead. Chimera introduces the idea that a version (or a configuration in Chimera) is not just a single object by any subset of documents and hypermedia objects. How the changes to the Chimera API carries over to the Chimera Web integration service is not described, but it must be assumed that additional work on Web integration needs to be done to support the versioned Chimera architecture.

The experiences that Whitehead sums up for Chimera neatly describes the problem of approaching hypermedia versioning by assuming that versioning can be handled disjoint by the document and hyperstructure services. The result is that synchronization between two versioning mechanisms is required – which, form an architectural point of view – is impractical and inelegant.

3.3 Hyperbases

While hyperbases, like HyperDisco [29], HAM [9] and HB3 [15], cannot strictly be considered "open" hypermedia versioning systems (the full integration of versioning is only possible if documents are stored inside the hyperbase) they have the great advantage of remaining in control of both documents and hyperstructure. Since versioning is a generalized

concept that spans both hyperstructure and documents the synchronization of the two version mechanisms is not an issue for hyperbases.

3.4 WebDAV

The storage facilities of the Web has until recently been quite primitive; limited to only read-only, unversioned access provided through Web servers implementing the HTTP standard. However, the work on WebDAV (www.webdav.org) has added both versioning; write access and document locking to the repertoire of the HTTP protocol, greatly expanding the range of document operation that can be done using HTTP. It is now possible to build a feature rich document management system directly on top of WebDAV. Nevertheless, the document format of the web, HTML, still requires that links and anchor are embedded with the documents. Embedding links allows only the owner of the document to add links and annotation – in direct conflict with the requirements of open hypermedia mentioned in the introduction. This limitation does not mean that WebDAV is a insufficient standard for open hypermedia. On the contrary, the distributed and scalable nature of WebDAV provides a strong foundation for hypermedia services. A distributed, versioned repository, such as WebDAV, is a natural companion to a versioned link service.

4. THE GAIA FRAMEWORK

Though several attempts have been made at creating versioning support in hypermedia, both for monolithic and open systems, no attempts have yet (to the authors knowledge) been made at creating a hypermedia versioning framework for the Web. While WebDAV strives to achieve versioning of documents on the Web, it does not provide an interface for versioning hypermedia objects. However, with the work on XLink this situation may be mended. Some research on creating add-on link and anchoring services for Web based HTML pages has been conducted (see [3] [7] and [8] for an overview). Nevertheless, adding versioning to these services is still an unexplored area –due to the size and distributed nature of the Web.

GAIA provides a framework and architecture for creating consistent versioning of hypermedia on the Web. It seeks inspiration in the ideas and work conducted by the Device Hypermedia Group [17] [18]. Devise Hypermedia is working closely with the established standards on the Web and integrates with technologies such as XML, HTTP and XLink. We must find a common interchange format for communication between open hypermedia frameworks, and existing Web standards inspire this format. In GAIA the following design goals are pursued:

- **An open Hypermedia repository built for the Web** – The docuverse is the entire Web and GAIA should be build to support its mechanisms. This includes awareness and compatibility with the underlying versioning mechanisms of the Web (WebDAV).
- **A separable open storage mechanism for hyperstructure** – Allowing links and anchors to be stored in their own separate repository, i.e. *not* embedded in Web documents. The access to this storage mechanism should be based on Web standards

- **Consistent versioning support for both hyperstructure and documents** – Meaning that all objects in the hypermedia framework can be versioned – this includes versioning of versions.

4.1 GAIA Architecture

The GAIA architecture consists of three services:

1 **The Docuverse Service** – Provides an abstraction level on top of the HTTP and WebDAV document storage of the Web. This abstraction level allows the programmer to see the Web as an unordered and unversioned collection of documents, each with their own unique ID.

2 **The Hyperstructure Service** – Provides an API for building and maintaining Dexter-like *unversioned* hypermedia objects. These objects include links, anchors and composites. The service refer to the docuverse service by pointing at the document ID's provided by this.

3 **The Versioned Navigational Hypermedia Service** – Building on top of the two others, this service provides the API for building open hypermedia versioning on the Web. It implements consistent versioning of both documents and hypermedia, mainly because documents and hyperstructure objects in the lower layers are unversioned seen from the version layer above.
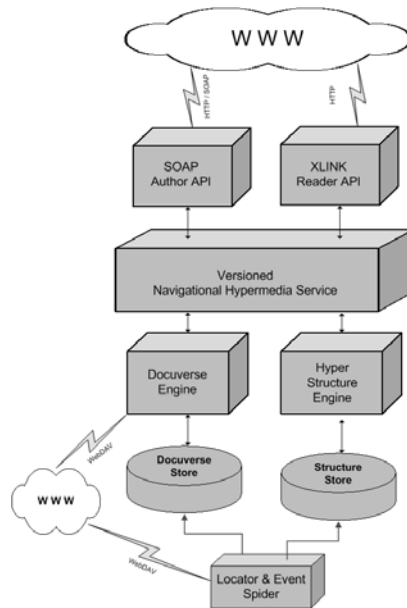

Figure 1. The GAIA architecture

4.1.1 The Docuverse Service

The Docuverse service handles the translation between document locations and "Document identity". While URI's provide the Internet with a location service it does not allow documents to be uniquely defined based on their identity. What then, is "document identity" - if it is not the document location? The notion of document similarity is a related concept: if every document in the docuverse is assigned a (globally) unique ID – it seems natural that two *similar* documents, although residing on different locations, should be assigned the same ID. We clearly need a service that defines the *semantics* of "similarity" in order to reuse the same document ID for different document locations. This requires maintenance of document metadata, which contains enough information to accurately compare the document with any newly discovered (i.e. referenced by the hyperstructure) documents so that the same ID can be assigned for documents that are *semantically* identical. The Docuverse Service maintains the metadata needed for comparing documents for similarity. It stores document ID's and information about where these documents can be found (URI path). Furthermore, it may be extended to provide a cache for documents that cannot be frozen (see 4.1.3)

For obvious reasons the service cannot, at least initially, reference all documents on the Web – but documents are added when they are first referenced in the hyperstructure. Since documents are given unique IDs, we can apply the abstraction of a flat (as opposed to a hierarchically ordered) docuverse.

4.1.2 The Hyperstructure Service

The Hyperstructure Service manages the storage and retrieval of all hyperstructural information in the GAIA framework. The hyperstructure contains: Links, Anchors, Composites and Queries.

- **Links** in the Hyperstructure Service can be either unidirectional or bi-directional, depending on their type. Each link has one or more endpoints, which contain references to anchors. In Whiteheads terms: the links *contain* anchors *by reference* [27].
- **Anchors** can point to documents, queries, and composites. Research in "location specifiers" [18] has shown that anchors may point at arbitrary locations within documents. Such an implementation is outside the scope of the project – GAIA deals with the hyperstructure itself, not how clients display anchors in documents. Whenever a link or anchor is changed, the service raises a change event. This allows the versioning service to track changes in the hyperstructure.
- **Composites** in the hyperstructure store are arbitrary collections of objects (incl. composites).
- **Queries** are small chunks of code that when interpreted (or "run"), returns a pointer to a subset of objects stored either in the Structure Service or in the Docuverse Service. In the current GAIA implementation a query is a named subset of the hyperstructure (much like a composite).

Each object in the hyperstructure is assigned a unique ID by the service – allowing objects to contain each other by reference only – as opposed to inclusion. None of the hyperstructure objects are version aware – version support is implemented *on top* of the Docuverse and Hyperstructure Services.

4.1.3 The Version Service

The Versioned Navigation Hypermedia Service (VNHS) depends on the Docuverse and Hyperstructure Services. Its primary purpose is to present an API for versioning both documents and hyperstructure.

The traditional approach to versioning Open Hypermedia has been to add versioning as a feature implemented inside *both* the Docuverse *and* the Hyperstructure Service. GAIA moves the responsibility of versioning to a layer on top of these. This is the motivation for implementing docuverse and hyperstructure services that present *unversioned* views of the objects that they manage.

A version in GAIA is *collection* of objects (including documents, other versions and hypermedia objects). Containment in GAIA is modeled by reference between version objects and their content. This means that any object can be a member of an arbitrary number of versions. In addition, versions can be members of other versions. This allows for a very flexible version model: The GAIA version model supports both multi level and cross document versioning. All objects are treated equally by the version service: both hyperstructures and documents can be versioned, even though the underlying layers of the architectures provide no version support.

4.1.4 GAIA Data Model

The GAIA version model is built on the principle of containment by reference. In order to compare the data model with other hypersystems we have used the containment models introduced in [28] – this model is seen in figure 2. Unlike most file systems, relational databases are not bound by hierarchical ordering of the stored data structures. Without the file system there is no need to model containment by inclusion.

All major objects in the GAIA framework inherit from a common object "GAIA Object" that provides basic functionality, such as Unique ID-generation, basic properties and freezing behavior (see 4.2). In addition the GAIA base object supports "reverse lookup" – i.e. it can answers questions of the form: "Which versions or containers contain me?" and "Which objects link to me?" – useful when traversing the network.

Version trees in GAIA are ordered collections of version objects – which in turn contain other GAIA objects. The version object in the tree is a special type of composite that implements methods to support common versioning operations (like deriving a new revision and branching/merging). Since versions are composites, a version may contain a version – which is what gives GAIA its multi-level versioning support.
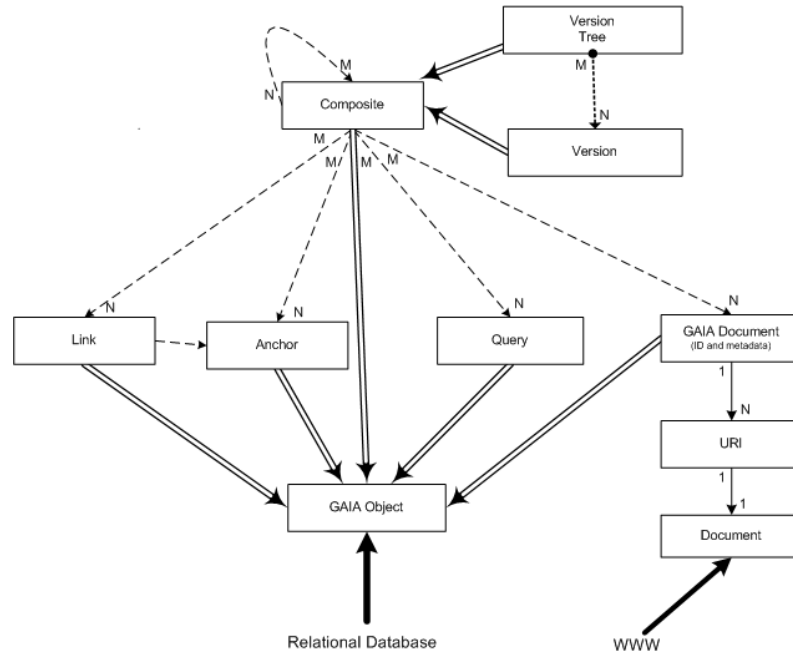
Figure 2. The GAIA containment model

4.2 Freezing Mechanisms

In order to support the construction of baselines a version framework must implement freezing mechanisms. In the hyperstructure service GAIA remains in control of the freezing behavior of hyperobjects – they are stored in a separate database that has been designed to support freezing. However, it is not possible to control the freeze behavior of documents on the Web (WebDAV only guarantee freezing if the web server guarantees that a checked-in document is never edited again), this must be compensated for in the Docuverse Service. If versioning mechanisms require that a document is frozen the docuverse service must do either:

1   **Copy the document at "freeze time" and save it in the Docuverse Service** –
    All future references to the document are to the copy and not to the original. The semantics of "copy" are not obvious though. A HTML document may refer to images, applets frames etc., which may or may not be considered a part of the document.

2   **Save a reference to the document location and save similarity metadata** –
    This does not guarantee immutability, since the author may either modify or delete the document at its location. Nevertheless, if the Docuverse Service defines the semantics of document similarity – we can detect that the document has changed. By using the metadata we *may* be able to relocate a document on the Web that is sufficiently similar to the one we were originally referencing.

27

Option 1 is a cumbersome task that may require our docuverse to store a rather big subset of the Internet. It is not an impossible task though: the Google web cache is an example such an approach.

Option 2, the approach used by GAIA, requires less storage space, but comes at the cost of lost references. This is however not new to users of the Web. If a document is moved or modified we are, as humans, able to determine that this is the case when we try to retrieve it. Search engines crawl the Web regularly to update their databases of document content, and we rely on them to relocate lost documents. Why then, should a hypermedia service not do the same? It may not be possible guarantee immutability – but it is possible to partially compensate by writing an automated agent that regularly crawls the docuverse looking for moved or modified documents. This approach will not eliminate the broken links – but with a proper notion of similarity (see 4.1.1) it is possible to either fix references automatically or notify the docuverse and versioning service that a change event has occurred, and corrections to document metadata needs to be made.

4.3 Example of use

GAIA is implemented on a Microsoft .NET platform and provides object libraries for accessing the version model it exposes. The libraries are implemented as .NET assemblies, and hypermedia structures are stored in a relational database. The GAIA .NET assembly can be included in applications that need access to a versioned hypermedia service.

Table 1. Excamples of creation of objects

```
// create some documents
 Document   M1V1  =  new   Document("Macbeth1v1.htm",
"M1V1");
 Document   M1V2  =  new   Document("Macbeth1v2.htm",
"M1V2");
 Document   M1V3  =  new   Document("Macbeth1v3.htm",
"M1V3");
 Document   M1V4  =  new   Document("Macbeth1v4.htm",
"M1V4");

 // Create some links and anchors
 Anchor A1 = new Anchor(M1Note1, "ANote1");
 Anchor A2 = new Anchor(M1Note2, "ANote2");
 Anchor AV1 = new Anchor(M1V1, "AV1");
 Anchor AV2 = new Anchor(M1V2, "AV2");
 Anchor AV3 = new Anchor(M1V3, "AV3");

 Link L1 = new Link("Link1");
 Link L2 = new Link("Link2");
 Link L3 = new Link("Link3");
 Link L4 = new Link("Link4");

 L1.AddSouce(AV1);L1.AddTarget(A1);
 L2.AddSouce(AV1);L2.AddTarget(A2);
 L3.AddSouce(AV2);L3.AddTarget(A1);
 L4.AddSouce(AV3);L4.AddTarget(A1);
```

```
VNHS.VersionedObject VO1 = New
VNHS.VersionedObject(M2V1, "Macbeth2");

 // fill the first revision with some structure
 VNHS.Version V1 = VO1.GetVersion("1.1");
 V1.AddContent(M2Note1);
 V1.AddContent(M2Note2);

 // derive a new revision and populate it
 VNHS.Version V2 = V1.NewRevision();
 V2.ReplaceContent(M2V1, M2V2);
 V2.AddContent(M2Note3);
 V2.AddContent(M2Note4);
 V2.RemoveContent(M2Note1);
```
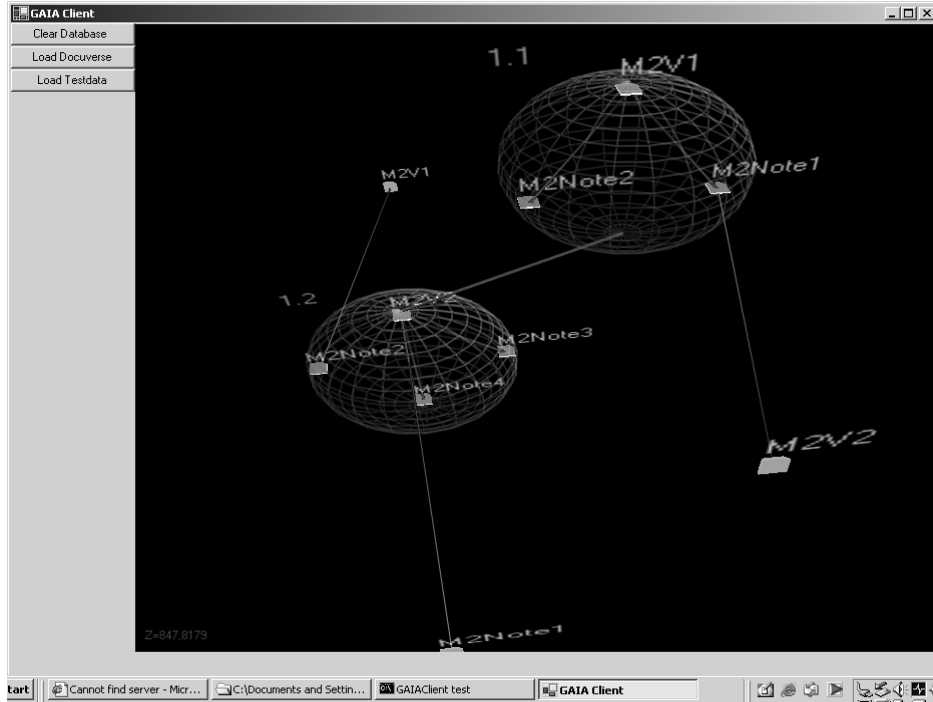
Figure 3. The object V01 inspected from the GAIA client

To test the GAIA framework a client application has been created that can inspect the versions and structures created by GAIA. The output of this application when inspecting V01 can be seen in Figure 3. To demonstrate the creation of objects in GAIA lets start by making a small collection of hyperstructure and docuverse objects in the small C# script shown in the left column of Table 1. The structures created so far have no version relationship to each other. The right column of Table 1 illustrates a few features of GAIA by modeling a versioning of structure.

A versionobject in GAIA is a structure for tracking a number of versions; it can be considered equivalent to a version graph containing (by reference) the objects of all the versions in the graph. The spheres are versions and black squares are documents. The spheres contain a set of objects. If we choose to freeze version 1.1 (the uppermost sphere in Figure 3) of VO1 the objects M2V1, M2Note2 and M2Note1 would be frozen. Version 1.2 (the lower sphere) links to M2V1 and the side effect of freezing version 1.1 will be that this link is frozen too. Since version inclusion is by reference it is possible for an object to be contained inside one version and outside (i.e. referenced by link) another version in the version tree. If Version 1.1 contained another version GAIA would freeze the transitive closure of the version containment relation. This is necessary if the entire state of version 1.1 is to be saved.

29

5. RELATED WORK

GAIA proves that it is possible to implement the vision of James Whitehead [27]: Using version objects that contain both documents and hyperobjects by reference. This adds strong versioning capabilities to the framework. By using Whiteheads version model GAIA achieves both multi-level and cross document versioning capabilities (see section 2.3). HyperDisco is the only framework with an open API that has thus far attempted multi-level versioning of a single object before. GAIA takes versioning one-step further by providing native support for multi level versioning, also of entire substructures of the hypernetwork.

The Chimera (3.2) and Microcosm (3.1) attempts at versioning hypermedia on the Web have both assumed that versioning is the responsibility or both the hyperstructure server and the document storage subsystem. Furthermore, the versioning mechanisms of Chimera and Microcosm rely on the capabilities of the underlying document store. GAIA takes a different approach and adds versioning as an add-on service, similar, and in addition, to the strategy of creating add-on link/hypermedia services. The open hypermedia community has already pursued such an "add-on strategy" extensively and it is natural to expand it to cover versioning. The only framework that has attempted such a complete version model before is the Xanadu project [22] – but Xanadu, though ambitious, cannot be considered an open hypermedia framework mainly because it relies on a proprietary document storage format: the "Xanalogical Storage".

A GAIA version can be any collection of objects, not just a single document or link. Chimera attempted such an implementation – but was faced with the problem of how to keep the versioned structure synchronized with the underlying document versioning mechanisms. Since GAIA moves the versioning responsibility away from the document and hyperstructure repository, version synchronization is not an issue.

Microcosm and Chimera are both inspired by the Dexter hypermedia model [14]. Considering the argued advantages of adding versioning as separable service instead of implementing it inside the document/hyperstructure storage, one may consider changing or expanding the recommendations of the Dexter model [14] to include versions as first class objects. Versioning should be considered part of any information system, and hypermedia frameworks assume the responsibility of providing versioning services.

GAIA builds upon the fast expanding WebDAV and HTTP standards – with the experiences from GAIA proposals for a few new features can be formulated:

- **Freezing** – There is generally a lack of freezing mechanisms for documents on the Web. Web servers should obey freeze request, or at least be able to return some indication of whether a freeze can be guaranteed or not. If freezing could not be guaranteed, the docuverse could choose to copy the document to a cache. However, the copyright implications, along with the massive storage demands, may make this approach less than optimal. Though Google does use a Web cache for its documents, it is still a big task to save a history of all documents on the Web.

- **Web server change log** – When documents move between servers – a change log would be of great help to users consuming these documents. When documents move, web servers should make more widespread use of the HTTP 301 status code to supply the client with information about document movements. This

would greatly help hypermedia systems like GAIA who rely on change events to maintain the version structure.

6. CONCLUSION

The prototype GAIA implementation has shown that it is possible to implement consistent versioning in hypernetworks. As suggested by Whitehead [27], implementing versioning as add-on service has proven a viable path to pursue. In GAIA, information can be related and retrieved in four distinct ways:

1   **Containment** – being contained within another object (including a version)
2   **Link** - connected through traditional navigational hypermedia links
3   **Query** – Retrieved as the result of a well-structured query.
4   **History** – Related through a version history

GAIA, like most hypermedia frameworks, provides direct support for the first three through the implementation of the hyperstructure service. Historical information is implemented by an external version service that maintains a consistent picture of the history (versions).

GAIA's flexible version model allows entire substructures of a hyper network to be tracked. Furthermore, the freezing problem (2.1) becomes manageable when the version model has the flexibility to support multi-level and structure versioning. The author can simply mark a given subset of the hyperstructure as a version and track the changes to any part of this structure with the framework. It is hard to visualize the link, containment and history levels at the same time. If versioned hypermedia is to be the method for managing large bodies of information, then we need to think of better representations of the structure. The "underlined link" only represents the link level; users must be able to identify containment and history relations too.

The flexibility of GAIA depends on events generated by the subsystems of the framework. Since GAIA remains in control of the hyperstructure it can auto generate events when changes occur in hyperstructure objects. Events generated by documents changing on the Web can be asynchronously detected by using an event spider service and storing meta data about the similarity of documents. Similarity in the GAIA framework is implemented as set of text feature vector of the documents referenced. Two documents are similar if the distance between them in small in the feature vector space. There are some problems with this approach though. Some documents, namely those auto generated by content management systems and ASP / PHP database access, cannot be considered "stable". The similarity information for these documents will never be really up to date, and chances are that change events will occur too frequently to be practical. Research remains on how these documents are to be stored by the docuverse service.

One of the central issues introduced by combining hypermedia and versioning is the freeze problem described in section (2.1). The only viable approach so far has been that a freeze of an object freezes the transitive closure of all links going out from the frozen object. By implementing versioning of entire structures, GAIA allows the hypermedia author to choose a limited subset of the hyperstructure to be affected by the freezing of a version. Using versioning of structure means that GAIA freezing is limited to the transitive close of the *containment* relation and not the link relation. Since it must be assumed that the containment

level for an object is rarely as deep as the link level (the cognitive overhead for this would be too great), GAIA makes freezing a more manageable concept.

This is no silver bullet though; hypermedia freezing makes it a necessity to transitively freeze the containment level of a versioning system. However, depending on the work area hypermedia is used in, some freezing side effects of the link level might still be desired, the semantics of which is left to the client.

The GAIA framework is a complete implementation of a hypermedia versioning framework and includes both a client application and a SOAP based API for accessing the Docuverse, Hyperstructure and Versioning service. Work remains on expanding the event systems (especially those of the Docuverse) to allow for better auto generation of new versions as a result of change events. In its current status GAIA provides inspiration for future versioning frameworks where versioning should be considered part of the initial architecture and not as features to be added at a later stage in the development process.

ACKNOWLEDGEMENT

REFERENCES

[1] Adobe System Incorporated, Portable Document Format Reference Manual

[2] Anderson, Kenneth M. et al.: "Chimera: hypermedia for heterogeneous software development environments", 1994, Proceedings of the 1994 ACM European conference on Hypermedia technology, September 1994

[3] Anderson, Kennth M.:"Integrating Open Hypermedia System with the World Wide Web", 1997, Proceedings of the eighth ACM conference on Hypertext April 1997

[4] Anderson, Kenneth M.:"Data scalability in open hypermedia systems", 1999, Proceedings of the tenth ACM Conference on Hypertext and hypermedia. February 1999

[5] E. H. Henderson, V. D. And Sigels S. G.: "Software Configuration Management: An Investment in Product Integrity", 1980, Prentice-Hall, Englewood Cliffs, NJ

[6] Bolinger, Don , et al: "Applying RRC and SCCS : From Source Control to Project Control (Nutshell Handbook)", O'Reilly publishers, 1995

[7] Bouvin, Niels Olof : "Designing open hypermedia applets", 1998, Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space—structure in hypermedia systems: links, objects, time and space—structure in hypermedia systems May 1998

[8] Bouvin, Niels Olof: "Unifying strategies for Web augmentation", Proceedings of the tenth ACM Conference on Hypertext and hypermedia, February 1999

[9] Campbell, Brad & Goodman, Joseph M.: "HAM: a general purpose hypertext abstract machine", 1988, Communications of the ACM July 1988, Volume 31 Issue 7

[10] Carr, L. A. et Al: "Link services or link agents", 1998, Conference on Hypertext and Hypermedia, Proceedings of the ninth ACM conference on Hypertext and hypermedia: links, objects, time and space—structure in hypermedia systems, 1998, Pittsburgh, Pennsylvania, United States

[11] Cederqvist, Per: "Version Management with CVS", 1992 Signum Support AB, at URL: http://www.cvshome.org/docs/manual

[12] Conradi, Reidar , Westfechtel, Bernhard: "Version models for software configuration management", ACM Computing Surveys (CSUR) June 1998, Volume 30 Issue 2

[13] Davis, Hugh: "To Embed or Not to Embed…", Communications of the ACM, August 1995, Vol. 38, No. 6

[14] Halasz, F., & Schwartz, M. "The Dexter hypertext reference. Proceedings of the Hypertext Standardization Workshop", 1990, (Gaithersburg, Md., Jalluary), pp. 95-133.

[15] Hicks, David L. & Leggett, John J. & Nürnberg, Peter J. & Schnase, John L.: "A hypermedia version control framework", ACM Transactions on Information Systems (TOIS) April 1998

[16] Durand, David: "Palimpsest: Change-oriented concurrency control for the support of collaborative applications", Boston University, Boston 1999, PhD dissertation, can be found at URL: http://cs-people.bu.edu/dgd/dgdDiss.pdf

[17] Grønbæk, Kaj & Trigg, Randall H.: "Design issues for a Dexter-based hypermedia system", Communications of the ACM 37, 2 (Feb. 1994), pp. 40-49.

[18] Grønbæk, Kaj & Trigg, Randall H.: "Toward a Dexter-based model for open hypermedia: Unifying embedded references and link objects", Proceedings of the the seventh ACM conference on Hypertext March 1996

[19] Halasz, Frank G. & Moran, Thomas P. & Trigg, Randall H.: "Notecards in a nutshell", CHI/GI 1987 conference proceedings on Human factors in computing systems and graphics interface 1987 , Toronto, Ontario, Canada

[20] Hall, Wendy & Hill Gary & Davis, Hugh: "The Microcosm link service", 1993, ACM Proceedings of the fifth ACM conference on Hypertext 1993, Seattle, Washington, United States

[21] Melly, Mylene & Hall, Wendy: "Version Control in Microcosm" in ECSCW 95, Workshop on the Role of Version Control in CSCW Applications, Stockholm, Sweden, 1995

[22] Nelson, Ted: "Literary Machines", 1983, Mindfull Press, ISBN: 0-89347-062-7

[23] Tichy, Walter F.: "Design, implementation, and evaluation of a Revision Control System", Proceedings of the 6th international conference on Software engineering September 1982

[24] Vitali, Fabio: "Versioning hypermedia", ACM Computing Surveys (CSUR) December 1999

[25] Microsoft Visual Source Safe Website at URL: http://msdn.microsoft.com/ssafe

[26] Whitehead, Jame E. Jr. et Al.: "A proposal for Versioning Support for the Chimera System", Proceedings of the Workshop on Versioning in Hypertext Systems held in ECHT '94, ACM European Conference on Hypermedia Techology, Edinbourgh, september 1994. URL: http://cs-people.bu.edu/dgd/workshop/whitehead.html

[27] Whitehead, James E. Jr.: "Design Spaces for Link and Structure Versioning", 2001, Proceedings of the Twelfth ACM conference on hypertext and hypermedia

[28] Whitehead, James E. Jr: "Uniform Comparison of Data Models Using Containment Modelling", 2002, In Proceedings of 11th International Workshop on Software Configuration Management (SCM-11), LNCS 2649, Portland, Oregon, May 9-10, 2003, pp. 70-85.

[29] Wiil, Uffe K & & Leggett, John J.: "The HyperDisco Approach to Open Hypermedia Systems", Proceedings of the the seventh ACM conference on Hypertext March 1996

[30] Wiil, Uffe Kock & Hicks, David L. & Nürnberg, Peter J.: "Multiple open services: a new approach to service provision in open hypermedia systems", Proceedings of the twelfth ACM conference on Hypertext and Hypermedia, 2001, Pages: 83 - 92

[31] Østerbye, Kasper: "Structural and Cognitive Problems Control for Hypertext in Providing Version", ECHT '92: European Conference on Hypertext Technology, November 30 - December 4, 1992, Milan, Italy