IUHM, a Hypermedia-based Model for Integrating Open Services, Data and Metadata

Marc Nanard
LIRMM, CNRS/Univ. Montpellier
161 rue Ada, 34392 Montpellier
France
Phone: (33) 467 41 85 17
mnanard@lirmm.fr

Jocelyne Nanard
LIRMM, CNRS/Univ. Montpellier
161 rue Ada, 34392 Montpellier
France
Phone: (33) 467 41 85 16
jnanard@lirmm.fr

Peter King
Computer Science Dept
University of Manitoba
Winnipeg, Canada
Phone: (1) 204 474 9935
prking@cs.umanitoba.ca

ABSTRACT

This paper discusses a new hypermedia-based model known as IUHM. IUHM emerged as a result of the development of the OPALES system, a collaborative environment for exploring and indexing video archives in a digital library. A basic design requirement of OPALES is that it must permit and support the integration of new services throughout its life cycle. Thus, IUHM depends heavily upon the notions of extensibility and openness.

Support for openness, extensibility and late binding of services is provided in the IUHM model by a single reflexive mechanism. This uniform mechanism is used for describing all relationships between arbitrary system entities, including services, data and metadata. The mechanism in question consists of a generic, computable hypertext structure with typed links, known as the *Information Unit,* and is the minimal structural scheme to which all encapsulated entities comply.

We describe and justify the design of the Information Unit, as well as the semantics of its four link types, namely *role, type, owner, relative*. We further describe the minimal kernel of the runtime layer responsible for the dynamic behaviour specified by the IUHM compliant hypertext network. We discuss the mechanisms involved in the dynamic binding of services and service composition. We illustrate these notions by real-world examples of the integration of metadata services within the OPALES system.

Categories and Subject Descriptors

H.3.7 [**Information Storage And Retrieval**]: Digital libraries - *systems issues, user issues.*

I.7.2 [**Computing Methodologies**]: Document preparation - *hypertext/hypermedia, languages and systems*.

General Terms

Management, Design, Reliability, Human Factors, Languages.

HT '03, August 26-30, 2003, Nottingham, United Kingdom.

Keywords

Hypertext structure, open hypermedia system, semantics, metadata, service integration, structural computing.

1. INTRODUCTION

Historically speaking, it has been widely recognized that a fundamental requirement of a hypermedia system is that it provide for creating and manipulating relationships between information items. During the past decade such relationships have undergone a multi-stage evolution within the open hypermedia community. Initial developments saw the provision of open link services among third party applications [2], in which interoperability issues were resolved by whatever ad hoc approaches were appropriate at the various levels [15], [19]. Such systems evolved first into open hypermedia systems [16], including hypertext domains [14], and then into component-based open hypermedia systems [14, 24]. This progression culminated in architectures based on middleware structure services [21, 20], and this ultimate stage has led to the emergence of a new field known as *structural computing* [13], which affirms the pre-eminence of structure over data.

It may be observed that the architectural models developed throughout this evolution all approach the question of interoperability by providing standard protocols and a standard interface between the different levels of components. While this approach has been entirely satisfactory as far as the hypermedia community is concerned, the question of interoperability does not disappear. Rather, questions surrounding interoperability are in part replaced by the necessity to provide a uniform back-end storage [23].

In this paper, we propose an alternative approach to the integration of open hypermedia services. Unlike the approaches just described, we accept heterogeneity and we do not seek a standard protocol, or interface. Rather, we introduce a new unifying scheme for integrating heterogeneous entities, including such diverse entities as data, metadata, services, user-groups, and ontologies. In so doing we apply the pattern *hypermedia as integration* introduced in [4]. However, we go further, and separate the *structure* of an entity from the *semantic* concerns relating to that entity, and, in a similar fashion, we separate the structure and semantics of a society of entities related by typed relationships.

In order to achieve this goal, we introduce the *Information Unit Hypermedia Model* (IUHM), a system integration model, which draws its inspiration from hypertext structures. In this model, all

entities (data, metadata, service, ontology, etc.) are represented in a uniform fashion, each entity being encapsulated as an *Information Unit* (IU). The model associates a minimal set of links with an IU. In particular, an IU has a *type* link and a *role* link, and these two separate links provide the distinction between *structure* and *semantics* in the manipulation of an information unit. Additionally, an IU has an *Owner* link, which specifies which entity may manipulate the Information Unit in question. The IUHM framework is both open and reflexive. IUHM supports structural computing in a manner similar to that described in [9, 13].

As a significant test bed, we have applied IUHM and its underlying principle of uniform management of heterogeneous entities to the development of the OPALES system [10]. OPALES provides a portal to a set of diverse open digital library services. OPALES is designed for the cooperative manipulation of shared multimedia documents among multiple users and user-groups. In particular, such documents may be enriched by multiple *annotation* and *indexation* structures through private and public workspaces. The various entities occurring in the OPALES system and their interrelationships were modelled as an IUHM network, thereby making OPALES and its functional core *reflexive*, as described in section 2.3.

The remainder of this article is organized as follows. Section 2 establishes the three fundamental architectural requirements of our framework, and section 3 discusses in detail IUHM, the hypermedia model. Section 4 explains the architecture model which we have used in OPALES for the integration of structural computing services, and further indicates how this architecture satisfies the requirements discussed in section 2. In section 5 we illustrate the application of the model to part of the OPALES system and in section 6, we discuss related work and conclude.

2. CONTEXT AND REQUIREMENTS

Many of the ideas to be discussed in this paper emerged as a result of the development and subsequent use of the OPALES system [10]. OPALES provides a collaborative user workspace for exploring and indexing video archives held at l'Institut National de l'Audiovisuel (INA), Paris. But OPALES is a digital library system which goes beyond the provision of simply indexed archival material: it provides semantic-based archival access. A fundamental design requirement of OPALES was to permit the addition of new services throughout its life cycle, and accordingly the design of OPALES depends heavily upon the concepts of extensibility and openness. We therefore developed the IUHM model and the corresponding architectural framework in which system development and maintenance, including the addition of new services, could be accomplished without the need to change either the overall system architecture or its data structure.

In order to make the integration of multiple services as simple as possible [1], the architecture must satisfy the following three requirements: it must be *open*, it must support *interoperability*, and it should be *reflexive*. In this section we first explain and motivate these three requirements, and we then indicate how each has been satisfied within the IUHM framework.

2.1 Openness

An *open* system is one in which the addition of services is both free and easily accomplished, in contrast to a closed system in which the available user services are fixed and pre-determined. The users of multimedia digital libraries frequently suggest new tools or new services which they would find helpful in the system. The system must be *open* to support the incorporation of such new services.

IUHM is an open architecture in this sense. More specifically, IUHM is not itself a user service, but consists rather of an open infrastructure. This infrastructure supports the integration of arbitrary services by providing a consistent and unified access method both for services and for data and metadata.

2.2 Interoperability

The term *interoperability* refers to commonality of access means for services in all domains, as distinguished from the provision of middleware components specifically related to particular domains. The term has been used by a number of authors and a number of approaches to interoperability are to be found in the literature. FOHM [8] for example, takes the approach of considering a limited number (three) of domains, and defining the semantics of operations across those specific domains.

The approach to interoperability found in IUHM is rather different. IUHM does not provide any explicit definition of specific data structures to be shared by applications. Rather, IUHM provides a unified mechanism which enables data and services to be mapped and interconnected. The mechanism has some similarity to an object-oriented approach, but there are significant differences. In particular, IUHM provides for global, large-scale system management, and makes use of hypertext structure to depict the actual resulting system architecture. Further, IUHM separates the data semantics from data structure and enables these two aspects to be described separately.

2.3 Reflexivity

The term *reflexivity* refers to uniform treatment of both the system and of all user-accessible items within the system. Reflexivity, and the consequent avoidance of specificity in architectural design, facilitate interoperability, as just described. Furthermore, the high degree of unification inherent in a reflexive architecture makes an open system simpler to design and to operate. Reflexivity is more a design choice, a solution, than a constraint.

The reflexivity offered by IUHM provides common treatment of all items in the system, whether such items be system *primitive* notions, or higher level entities such as *services, data, meta-data*. In particular, reflexivity implies that a user can handle all items in such different domains in a uniform fashion, and therefore reflexivity provides support for our approach to interoperability.

2.4. The IUHM approach

The approach used in IUHM to meet the three system requirements just described is based on two main considerations.

Uniform hypermedia model. We rejected an approach which adopts distinct models for describing hypermedia structures on the one hand and services on the other. The use of such an approach in our view, complicates the management of openness and interoperability while maintaining homogeneous semantics [16], [7], [23]. Therefore we have designed a single hypermedia model IUHM, which makes use of a uniform reflexive data item known as the *Information Unit* (IU). All system entities of whatever type are encapsulated as IU instances. Such entities include *primitive notions*, *services* (such as indexing), *data* (such as digital video

archives), *meta-data* (such as conceptual graphs and ontology descriptions), *collaboration issues* (such as user groups descriptions, viewpoints). The information unit represents a hypertext node and the model includes a limited set of typed links which are used to depict a specific system in the large as a network of information units connected by structural and semantic relationships. Accordingly, all data items have a similar structure and are managed in a uniform manner. The IUHM model expresses structural aspects and is presented in detail in section 3.

Generic Functional Kernel. We have designed an open component-based infrastructure which offers generic and interoperable access and execution mechanisms. The infrastructure is based upon a generic functional kernel, which relies on IUHM. In order to bootstrap the reflexivity, any entity in the functional kernel is handled as an information unit. The infrastructure is *open* since it specifies no predefined data structure, semantics or behaviour for items beyond the primitive information units. The kernel provides the syntax rules that the IU must conform to; the semantics of an IU are defined by its own management rules, not by the structure of the data which it transmits. The infrastructure components and its functional kernel are described more fully in section 4.

3. THE INFORMATION UNIT HYPERMEDIA MODEL

This section presents in detail IUHM, the Information Unit Hypermedia Model. IUHM is not itself a model for hypermedia, but is rather a system integration model based on a hypermedia model, which uses structural computing techniques [9]. IUHM is based on a particular node structure, known as the *Information unit,* IU. We show how this structure unifies the notions of service and data, so that every item in the system is handled according to the same basic rules. Relationships between entities in the system are expressed by means of a hypertext data structure enabling structural computing. This structure facilitates the dynamic construction of tailorable end-user systems. Further, it becomes possible to edit the system structure in the same way as any hypertext data structure. This *reflexivity* makes a system more robust and easier to build and to customize, as we illustrate in section 5 in the case of the OPALES system.

3.1 Information Units

The IU provides a mechanism for the explicit and efficient management of semantic relationships between hypertext nodes. This management is performed by means of a hypertext with typed links structure [11] making use of what is known as structural computing. In order to distinguish the link structure which describes the system architecture from other links used for user-level navigation, IUHM introduces the notion of *descriptor*.

Considered as a hypertext node, an information unit is structured as two parts, its *descriptor* and its *content*. The descriptor part provides answers to four questions concerning the IU, namely *how* is it used, *what* is it for, to *whom* is it connected, and *who* can access the IU. Accordingly, the descriptor comprises four link fields, respectively

- *type*: the kind of data contained by the IU;
- *role*: the semantic purpose of the IU;
- *relative*: specification of relationships among IUs;
- *owner:* access rights on IUs;

Each of these four fields is implemented as a bi-directional typed link to an IU which contains the corresponding type, role, etc. Moreover, in accordance with the notion of to structural computing, each of the type, role, owner, and relative of an IU, is characterized not simply by the link or the destination IU, but rather by the whole link structure in the vicinity of the IU. The content of an IU stores the information itself. Figure 1 illustrates the information unit, the four link-fields comprising the descriptor, and the content; it should be noted that the descriptor and the content may, if convenient, be physically disjoint, since an IU is always accessed by its descriptor.

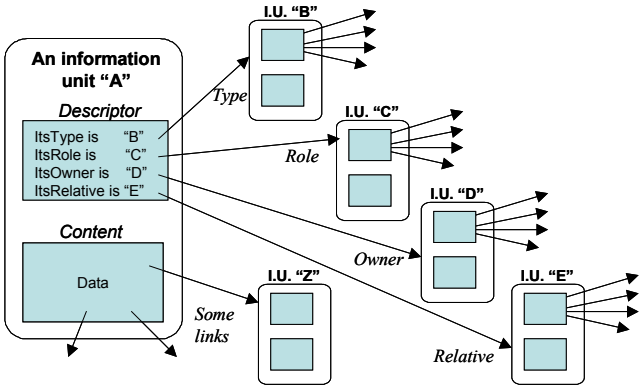

Figure 1. Hypertext Network of Information Units.

Fuller details of these four link-fields are given below, but for now we note that a system developed using this approach is fully specified by a hypertext IU network induced by these links. The IU descriptor also includes data such as *unique-identifier*, *name, access rights*, *access counters*, and the like.

There is no restriction on IU *content* data-structure. Such content can be simply a piece of text or a XML document or even a piece of code as in the case of a service IU. Hypertext links towards any other IU may also be found in the content part; these links are not specified in terms of the model and permit normal hypertext navigation.

3.1.1 Types and Roles

IUHM makes an important distinction between the *type* and the *role* links of an IU, a distinction not generally found in classical object-oriented approaches.

This distinction enables a separation of technical aspects and semantic nuance. The type of an IU characterises the data structure of the IU content, regardless of its usage. More precisely, the type link references the information unit which models the technical aspects of all information units having that type, thereby giving access to the primary level of software capable of handling the content, as shown in the example of figure 2, which is taken from the OPALES implementation.

The role of the IU characterizes its semantic behaviour independent of its type, and references the information unit which models the semantic behaviour of all information units sharing the same role. Role and type are independent: two IUs may have the same type, both are XML files say, but may contain quite different kinds of information, and thus have different roles. As a simple example, the role might be *user group description* and the type *xml*.

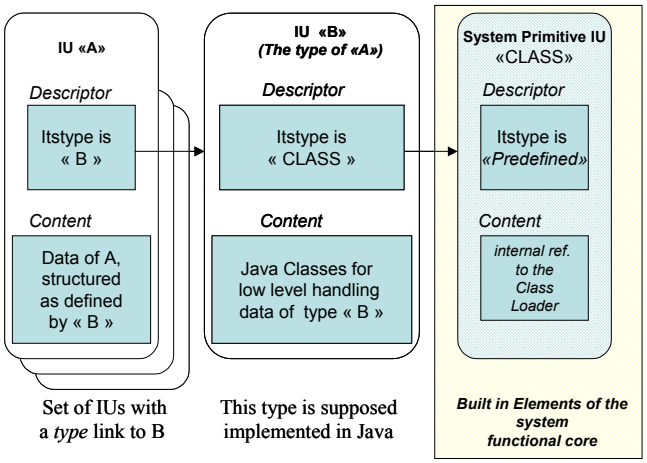

Figure 2. Example: Hypertext Implementation of Information Units in OPALES.

3.1.2 Primitive Information Units

As is the case in any reflexive framework, IUHM bootstraps from a small set of primitive information units, including the primitive notions used in the system. To this end *primitive* information units, such as EMPTY, UNDEFINED, PREDEFINED, CLASS, SYSTEM are defined as nodes within the structure. For example, the role link of an IU whose role is as yet undefined, is not a dandling role link, but references the *primitive* IU named UNDEFINED whose owner is the SYSTEM, whose type is PREDEFINED, and whose owner is ANYONE. In this fashion all relationships between items in the system are fully described by the hypertext network built out of these four links.

We now describe in greater detail the rationale for and the mechanisms associated with each of the four links within the descriptor of an IU. Section 5 illustrates a novel use of these basic mechanisms in the development of general services for collaborative annotation and indexing by interest groups.

3.2 Types and IU Content Manipulation

The type of an arbitrary IU a, say, is by definition the IU b, say, which is referenced by the *type* link of a. Further, the IU b provides in its content part the code necessary for handling the content part of the IU a. The principle of reflexivity implies that these two rules apply to all IUs within the system. We consider two contrasting examples. First, in the extreme case, of a primitive type, that is a type such as TXT or JPG which a system can handle without added tools, the type link references the IU PREDEFINED. In the more general case, the primitive IUs CLASS and CODE denote, respectively, downloadable and resident code segments. Thus, the Java classes of a service are found in the content part of an IU whose type is CLASS and whose role is SERVICE. As a direct consequence, the hypertext network always provides the proper code for low level handling of the data of any IU insofar as its type link references this code. The approach is entirely reflexive, and indeed the IU named CLASS has type CODE, since it corresponds to the built-in class-loading feature.

This general approach provides a simple, extensible mechanism, without putting constraints on the kinds of data that could be handled by the system. IUHM compliance implies that any IU is linked to its appropriate code. Adding a new data structure is as simple as adding new data, and in both bases one stores the classes which handle the entity in question in the IU content, and characterises this IU by its role and type links. Furthermore, this technique automatically adjusts the low-level processing of data within a generic service. For instance a general service whose role is to handle, say images, automatically uses the proper low level code to handle the content part of a given IU since the code is linked to the IU.

3.3 Managing IU Ownership

Regardless of its type or role, every IU has an owner, specified by its *owner* link, which is responsible for its creation and subsequent management. An IU owner may be, say, a user, a group moderator, the system administrator or the system itself, all of which are represented as IUs referenced, as appropriate, by IU owner links. The primitive IUs ANYONE and SYSTEM are included to support reflexivity. At one extreme, an IU whose owner is the IU named ANYONE may be handled by all users. At the other extreme, an IU which has the IU named SYSTEM as owner, is hidden to any users, but available to the kernel. This mechanism is used throughout the system, and indeed provides the basis for the management of user and user-groups and workspaces.

3.4 Roles and Objects

Roles permit a semantic structure to be induced on IUs, which is independent of their type structure. Consider, as an illustration, the *annotation* of, say, an item in a video library. An annotation may be of a variety of types. One may create an audio annotation, a graphical annotation, a textual annotation, or a more formal annotation using conceptual graphs, or even use RDF. However, the view of an annotation, as a metadata anchored into a document is independent of the type of the annotation. Accordingly if one considers the set of IUs which would be used to represent the above list of annotation types, each IU would have its distinct type, but *Annotation* would be used as their common role.

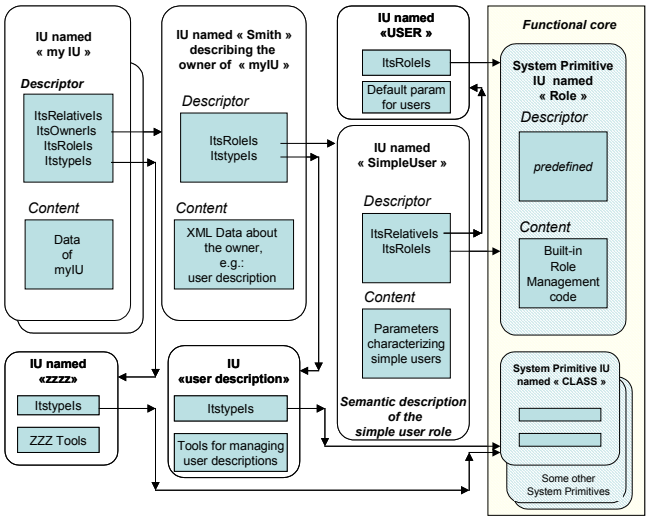

Figure 3. Example: Roles, Types and Owner in OPALES.

Figure 3 contains an illustration of these concepts. The Owner link of the IU *myIU* references a node describing the owner of *myIU*, the IU named *Smith*. Such an owner node is semantically characterized by its role link, which references an IU named,

here, *SimpleUser* whose role is to characterize the semantics of a simple user, i.e. what a simple user may do. To change *Smith* into an administrator it suffices to link its role to *administrator*.

The type link indicates simply that this data is described in XML, and provides access to generic tools applicable to XML files. An IU referenced by a role link is said to be a *role*, and its Role link references the primitive IU ROLE.

3.5 The *relative* Relationship

The *relative* link of an IU permits the definition of an arbitrary, directed relationship between an IU pair, and thus serves to construct a free and rich structure over a set of IUs. The relationship between two IUs designated by *relative* is often one of dependency, delegation, or inclusion. There is no predefined semantics for the relative link in IUHM; its interpretation is usually delegated to the role. For instance, in OPALES, the IU relative to an *annotation* IU is the annotated IU. The IU relative to an IU referenced by a *type* link is the IU to which it delegates. An analogous mechanism is used to implement role delegation and role hierarchies.

One might ask why IUHM has a single relative link. On the one hand, this restriction makes the model far simpler to handle. On the other hand, multiple relative links are infrequent and are simple to construct. Multiple links are implemented as a reference to an IU whose content is a set of links. This approach has been successfully used in the development of OPALES. For example, an IU whose role is to be the *answers* to a query has a content which links to the selected IUs. Such IUs are used for handling persistent or temporary storage of answers. In the same manner, an IU *slides* may be referenced by the content part of an IU *slideshow*. We observe that this feature was requested by end-users of OPALES, and is considered by them to be helpful.

4. ARCHITECTURE AND DYNAMIC ASPECTS

The previous section discussed the static structural aspects of a IUHM compliant hypertext network. This section focuses on the dynamic aspects of open service integration in a IUHM network. The reflexivity of IUHM means that the relatively simple techniques may be used. We use the term *functional core* to denote the minimal set of features to be implemented in a hypertext engine making use of IUHM. In terms of the Dexter Hypertext Reference Model [5], most of the functional core is embedded in the *run-time* layer of the hypertext engine, and the *storage* layer consists mainly of a IU server. The *within-component* layer consists for the most part of services within system components, although a service may be far more complex than, say, a simple component presenter.

Whereas the infrastructure developed for managing a IUHM network was developed independently of any IUHM compliant system, the functional core of the OPALES system is taken as an example to introduce these notions in a practical context.

4.1 Open Services

Let us first clarify the notion of open service in a hypermedia system. The *within-component* layer of the Dexter model is usually concerned with handling a single node data structure. A service in our sense is a more powerful notion, encompassing not only single-node processing, but also processing over multiple nodes, including the entire hypertext network itself. Thus, searching, annotating, and indexing are services, but so too are more complex operations, such as organizing one's bookmark space, or preparing a virtual slideshow-like presentation by transclusion [12] of anchored parts of hypertext nodes. Such a complex service can be built as a composition of other services. Moreover, the system is reflexive, and therefore processes which involve services, including the system itself, are also regarded as services.

This generality enables a hypertext system to be bootstrapped from its functional core by integrating the services which constitute the system. IUHM makes it possible both to store the actual code of these services directly in the hypertext nodes, and to specify the relationships needed by the functional core in order to integrate these services in the system. Our experience in using this approach in the development of OPALES has shown it to be both useful and efficient.

4.2 Overall Architecture

In figure 4 we present the infrastructure model to which the OPALES system complies.

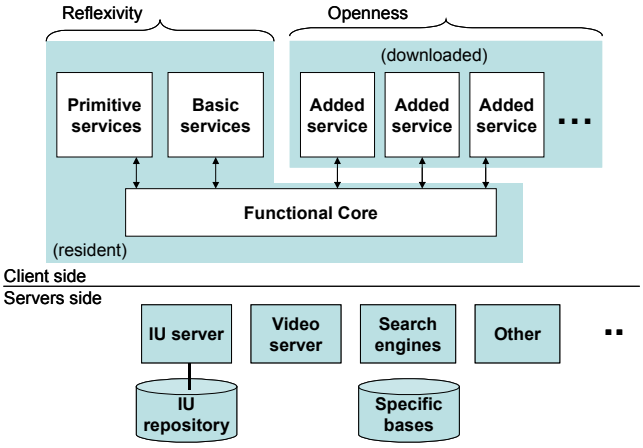

Figure 4. The Infrastructure Model.

At its simplest level, the architecture may be regarded as a client-server system in which the server side acts as the storage level of the hypertext engine. An IU server handles the IUHM compliant structure, and is responsible for the management of descriptors. Contents, for their part, are located in specific servers as required, including the IU server. For instance, in OPALES for reasons of efficiency video archives are stored on a dedicated video server. Conceptual graphs, which are used for semantically rich indexing, are stored on a specific server close to a *conceptual graph search engine*. Conversely, the content of a user private annotation is stored on the IU server, which serves as a private workspace repository for the OPALES users.

On the client side, the functional core, implements the runtime layer, whose purpose is to support the integration of arbitrary services, which are themselves described in terms of IUs. Thus, the functional core mimics the behaviour of a mother-board bus, providing a consistent and unified access both to services and to data and metadata, and providing a generic interface for creating, searching, accessing and updating IU descriptors and IU content. The functional core administers the dialog with servers, so that services reference data or other services in terms of IU, regardless of their actual location. In this fashion the internal dialog is

expressed in terms of properties of IU descriptors in the IU networks

The separation of descriptor and contents, at both the model and the implementation levels enables very fast navigation within the structure. Further, the reasoning done within the functional core depends upon structural computing on the IUHM compliant hypertext network.

4.3 Mapping Services to Data

As described in section 3, the type of a IU provides access only to the low level code for handling the IU. The functional core offers four categories of functions:

- *Get descriptor*: delivers a set of IU descriptors matching some selection criteria.
- *Open*: selects the relevant service to open an IU and triggers opening of the IU.
- *Select service*: gets the best-fit service according to some property.
- Primary access functions such as *Get content*.

The provision of such generic features by the functional core, with respect to a IUHM compliant hypertext, avoids the need for service code-segments to be hard-wired.

At a higher level, service selection derives from a dynamic mechanism embedded within the functional core. Since services are dynamic objects their run time presence in the functional core must be *registered*. In order to facilitate the selection of an appropriate service, such registration includes the set of signatures, in terms of the types, the roles and the owners, characterizing the IUs which the service can handle. This set is referred to as the *capacity* of the service.

Once a service is loaded in the functional core and registered along with its capacity, a service can be accessed in three ways:

- using its explicit name, or its unique identifier,
- using its IUHM descriptor properties as an abstract signature,
- using its dynamically registered capacity.

As an example, when opening an annotation in OPALES, that is an IU role-linked to the IU *annotation*, the core dispatches the service request to the most specific service currently able to handle it, according to criteria within a set of services. Since the annotation IU has a type, the selected annotation service may dynamically access the data to be annotated by using code for low-level handling of IU content data.

4.4 Generic Service Invocation

The mechanisms for adding and for using services are unified within the functional core by virtue of the general IU access management mechanism. One service can access a second service as an IU and thus services may dialogue with each other in a functional manner. A service can retrieve another service by name or by any property which accesses the appropriate IU.

In order to initiate a dialog with another service, a service requests the functional core to provide a reference to the other service object in memory. This causes the service[1] to be downloaded from the server (if necessary), and to be launched.

Any service can request the processing of a given IU by a specific service, using the *open IU* command and specifying the required properties of its partner. Generic composition of services is easily handled using this mechanism.

Consider as an example the service *answer-presenter* within a search engine, which presents the answer to a given query. Rather than itself opening the IU corresponding to the successful search, answer-presenter may decide to give access to a second service *IU inspector* to display information concerning the retrieved IU. To achieve this, the search engine simply makes a request to functional core to open the IU with a service whose role is *inspector*. Since several services may have registered as inspector, the core will assign the best match for the IU to be inspected. In this way if an appropriate inspector has registered for handling the selected IU type, such an inspector will open and inspect the IU. It is not usually the concern of the caller to decide the specific tool which will be used in any such case, although a caller may, if required, set strong constraints on the selection of such a partner service.

4.5 Service Composition

The mechanism described in the previous paragraph can be used to construct complex services by integration of several other virtual services. The IUHM architecture enables composite services to be specified dynamically in terms of their functionality rather than by hard wiring actual components.

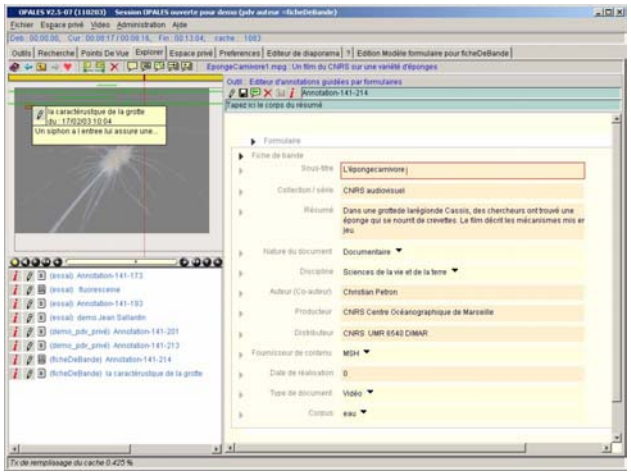

Figure 5. Service integration: VideoExplorer, an annotation editor, and a selector.

As an illustration, OPALES provides a service to *annotate* an explored document, that is to link other documents to it. The annotation service integrates three *virtual* services: an *explorer* service, an *annotation-editor* service, and a *selector* service. When an annotation is accessed, a relevant annotation editor is selected with respect to the IUHM hypertext network which depicts the IU properties. The appropriate editor is dynamically assigned to the generic slot in the compound service. Similarly, the annotated document IU which was bound to the annotation by

[1] or its local delegate, if the service code runs on another server, as is the case for OPALES conceptual graphs search engine.

the relative link, causes the assignment of the associated explorer service. The anchorage of the annotation in the annotated document causes the selector to display a list of other annotations whose anchors intersect the current anchor, enabling the user to browse other annotations pointing in the same vicinity. This is operation illustrated in figure 5. The operation of this compound service is simply to manage the co-operation between an explorer, an editor and a selector service.

Since the assignment of IU to services is handled by the functional core, service integration is specified in a functional manner rather than in an ad hoc fashion. Deciding upon the programming interface of a selected services is not the concern of the IUHM system, whose function is rather to assert that the loaded service is registered as an editor, and had the capacity, as defined earlier, to handle the proposed content data structure.

A compound service can itself be virtual. As an illustration, a *slideshow* service has recently been added to OPALES. *Slideshow* and *Slides* are both IUs, and associated editors and explorers have been created for their manipulation. In this way a compound virtual service is derived from the annotation service, which handles *slideshow* related IUs rather than *annotations*. The slideshow overview explorer is automatically assigned to the generic explorer slot and the slide editor to the generic editor slot. As a consequence, the slideshow overview is automatically presented to a selected slide, as its *relative* document, and the *selector* provides direct indexing into the slide show.

4.6 Service Downloading

Service downloading is a feature which relies directly on the general IU access management within the functional core.

By virtue of the reflexive nature of IUHM, the *openIU* command works in exactly the same manner for services as for data. The entire system is built up by using a simple bootstrap. A predefined service *service installer,* which makes use of the Java run-time class loading feature, is built-in to the functional core. The service *service installer* is registered itself as a service handling IU with role *service* and type *class*.

At session start up, the client requests the IU server to get the IU descriptors of services that are visible to the current user. Whenever the user decides to use a service, or when data processing by another service requires it, if the service is not yet loaded in the local cache, an open IU command is issued for the service in question. The general mapping mechanism dispatches this IU to the best fit, which is in this case the *service installer.* The *service installer* gets the IU content from the server, stores it locally as Java classes, and instantiates it. The loaded class must declare its capacity, and is responsible for its own protocols. The only interface specification is that the constructor of a service class calls the methods for registering the service in the core. Thus, there is no predefined constraints on what data or what functionality may be supported by the system. The IUHM architecture assures the correct mapping between data and tools and the composition of services.

4.7 Open Service Architecture

Since all entities, services, data, metadata, users, user groups, and so on, are unified in terms of IUs, the same mechanisms are available to index, describe, retrieve, or download any type of entity. Whether an end-user is searching for a document, for a service, for another user, or for a v*iewpoint* [10], the search will be conducted in the same manner, using the same tools. Similarly, services operate in a functional manner, and may operate recursively on themselves. Thus, one can index a service, describe a service in an annotation, etc. It has long been recognized that this property, known as *uniform referents,* is helpful to the user and provides for a simpler user interaction scheme in such an open environment [16] [3].

IUHM provides a minimal framework for supporting openness. Openness usually requires strong protocols in order to ensure consistency. Such protocols are not the primary concern of IUHM, but the IUHM basic mechanisms provide support for protocol checking. Constraints, protocols can be attached to any IU in a same way that annotations may be attached to services and to interface description by an end-user. IUHM provides structural rules to manage an executable specification, the functional core of the run-time layer provides tools and mechanisms to handle such a specification, but neither IUHM nor the functional core determines the semantics of how these rules will be used.

5. MODELING NEW SERVICES – AN EXAMPLE

This section illustrates how flexibility on the end-user side is supported by an IUHM compliant hypertext, and how a new service may be added to a system.

We base our description upon an example taken from the existing OPALES environment, and we discuss the addition of user group management as a new service.

Digital libraries provide the opportunity to go beyond the provision of simply indexed archival material and to permit semantics-based archival access. OPALES permits the development of generic, open services for the accumulation of large amounts of individual annotation and indexing effort, and for the creation of a community management mechanism, which permits users to share elicited knowledge. The notion of *viewpoint* has been introduced in OPALES to provide support for such facilities. Viewpoints support the management of small knowledge clusters specific to user communities. Viewpoints are fully discussed in [10]. Here we discuss how viewpoints provide generic and flexible services for semantic interoperability through the management of *interest groups*, which share elicited knowledge in the form of annotations.

5.1 Basic Annotation Mechanism

Figure 6 depicts a video archive, described by an IU whose *owner* is the institution responsible for archiving the video, in this case l'INA. The *relative* link references the archive category in which the video is stored in the institution. The *role* of the IU is to be a video archive while the *type* specifies the video format used and designates the corresponding software. The contents of the IU is the digital video on the archive video server.

A user who accesses this video archive IU may display and *explore* the archive, since these functions are provided by the role ARCHIVE, but the user cannot edit the archive unless she is the owner, that is, the archivist. She may, however, *annotate* the video archive, since the action *annotate* is supported by the role ARCHIVE. When clicking on an *annotate* button, the explorer creates a new IU with role ANNOTATION, anchored to the current selection. In this way, the user gains access to the annotating software provided by the type of the annotation.

Assuming that a video-segment was selected for annotation, this segment becomes the *relative* of the annotation. The annotation *owner* is the only user authorized to set the access rights on annotation, and make it visible and editable by others. However, other users who access this annotation may further annotate it, since *explore* and *annotate* are actions supported in the annotation role.

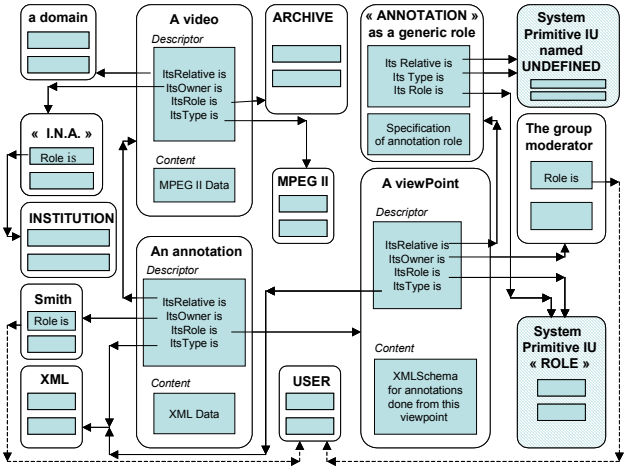

Figure 6. Annotations and Viewpoints.

5.2 Interest Group Driven Annotations

Suppose now that the user belongs to some specific interest group, say she is an ethnologist, and that she wishes to annotate the video as a member of this interest group. The role of annotations created on the behalf on a given group is called the *viewpoint* of this group. *Viewpoint IUs* form a specialization of *annotation IUs*. This hierarchy is depicted by the *relative* link, see figure 6.

A viewpoint IU represents an interest group as an ANNOTATION role. When a user indicates in the environment that he is authoring on behalf of some interest group, the annotation service assigns the group viewpoint IU to the virtual annotation role of the explorer service. In this way, when the user creates an annotation the actual role of the created annotation IU is the VIEWPOINT representing the interest group. Since any annotation made on behalf of this group has the same viewpoint as its annotation role, the editor dynamically chosen by the functional core for creating or opening these annotations is selected according to this viewpoint. Declaring such properties enables an interest group to set and enforce its own rules and the services to be used in this context. The templates for annotating, the associated ontology and the like can be constrained to conform to group standards.

5.3 Managing Viewpoints

A viewpoint is an IU like any other, and thus has an *owner* who created it and has sole responsibility for editing it. The owner acts as the moderator of the user group associated with the viewpoint in question, and can set the rules for indexing. The owner of the viewpoint gains access to the viewpoint editor service. The viewpoint contents, as an annotation role IU, depicts the semantics of the viewpoint and its associated tools and data.

Suppose now that the moderator wishes to use the default XML editor registered for annotations, but with a specific indexing template. In this case, the moderator just has to edit the XML schema of the indexing template which has the viewpoint *relative*. Conversely, if the moderator prefers to use a specific editor instead of the default XML editor, she just has to adjust the desired editor to make it register for this viewpoint role.

5.4 Multi Viewpoint Search Service

One of the originality features of OPALES is the ability to organize annotations into viewpoints and thereby to offer a multi viewpoint search service. People who search in television archives using OPALES can mix points of view within a compound query. For instance one search a document about China, say, using some criteria from a historical viewpoint, and some other criteria from a medical viewpoint. Each viewpoint is freely and independently managed by its own moderator.

Typically, the multi viewpoint search service is a dynamically compound service, since each part of the query is expressed using the specific editor associated with the selected viewpoint and with the associated ontology. Selecting a viewpoint in this multi viewpoint search service causes the functional core to dynamically load the relevant query editor along with its context. The compound service task is to combine appropriately the elementary results produced by the specific tools. Adding a new viewpoint does not imply any change in the multi viewpoint search service, since the mapping of tools to data is done dynamically by the functional core in accordance with the IUHM structure.

6. CONCLUSION AND RELATED WORK

In this section, we summarize the main contributions made by IUHM to the issues of openness, interoperability and reflexivity for providing multiple open service integration for digital libraries. We discuss these contributions in relation to other published work. We conclude with some future directions for our own research.

6.1 Summary

IUHM draws its inspiration from hypertext structures but is not by itself a hypermedia model. It is rather a system integration model based on hypertext structures. The IUHM framework provides generic means to interpret IUHM compliant hypertext structures. The main contributions of the IUHM framework are as follows:

- The *reflexivity* of the IUHM model provides a generic and homogeneous manipulation scheme by encapsulating any system entity, data as well as services, using a single mechanism, the *Information Unit*.
- The distinction between the *type* link and the *role* link permits the separation of structural concerns from semantic concerns.
- The *functional core* offers a generic execution mechanism as a minimum run time layer for interpreting the IUHM network according to user navigation, thus providing *interoperability* and *openness*.
- IUHM provides *structural rules* to manage an executable specification. The functional core of the run-time layer provides tools and mechanisms to handle such a specification, but neither IUHM nor the functional core determines the semantics of how these rules will be used

The OPALES system was developed on top of the IUHM infrastructure. Most of the OPALES client has been developed in Java. The IU server is a set of servlets running on Apache Tomcat, jointly with MySql. The video server, as well as the Conceptual Graph tools and ontology management tools have been written in C++. The java code of Opales client is about 60000 Java lines, the part of the functional kernel which implements IUHM is about 3000 Java lines. About 120 general purpose predefined IU are used by the functional kernel and about 80 by OPALES itself. Other IUs result from OPALES activity.

We have also illustrated in the article, the usefulness of the IUHM mechanisms to model and implement asynchronous and implicit collaboration services in OPALES. OPALES is currently in experimental use on several video archive corpus. We are able to test and evaluate different metadata services and research services, as well as the collaboration service at La Maison des Sciences de l'Homme (MSH) in Paris.

6.2 Discussion

Our work may be discussed under the following headings:

6.2.1 CB-OHS architecture

The architecture of current Component-Based Hypermedia Systems (CB-OHS) for integrating multiple services [24] pre-supposes a hypermedia model capable of offering a common structure model to different hypermedia domain services and more generally to structural and other computing services with a common semantic interpretation [7]. The architecture is organized in three levels: the level of services is distinguished from the level of client applications and data, and the level of backend hypermedia storage. This separation makes it necessary to provide standard protocols to manage interoperability of access to services or between services and the backend storage [23]. It appears difficult to use a third party application as a secondary tool associated to a service, say a third party editor application with a metadata service to produce metadata. Furthermore cooperation among services requires a common semantic interpretation by such services of the generalized hypermedia model. FOHM [8]. was designed with this aim, but the approach used in IUHM and OPALES is different. Our approach to the problem of interoperability at any service level resembles the approach in [22] for high-level specification of services. IUHM accepts heterogeneity and provides a common encapsulation mechanism, the IU, together with a common execution scheme. Furthermore, the IUHM model does not impose a particular operational semantics for services; such semantics may be defined by the services themselves. The reflexivity of the IUHM model provides a powerful means for composing services, which, together with the generic execution mechanism of the functional core, provides openness for the addition of an arbitrary new service and new usage policy. The need to introduce a common execution service is referred too in [18]. In contrast to the approach of [20], OPALES does not need to distinguish between horizontal (intra layer) and vertical (interlayer) interoperability of services or to consider common semantics of abstraction translation To resolve these issues, OPALES uses encapsulation, and distinguishes structural interoperability and semantic interoperability by means of the distinction between types and roles.

Discussion of openness of services in the OPALES infrastructure, in the precise sense of [20] is beyond the scope of this paper.

6.2.2 Programming in the large

The description of component-based architecture is presently a hot topic in software engineering.

At first sight, it may appear that the IUHM architecture we have described has much in common with the object-oriented approach. While there is some degree of similarity, the differences between these two approaches are of far greater significance. Both IUHM and the object-oriented approach support late binding. However, in the case of an object-oriented language such as Java, such binding is restricted to Java objects in persistent storage, whereas in the IUHM model, such binding may be applied to *any* type of object. This generality is, of course, exactly what we mean by openness, and is at the heart of what we have set out to do. It is precisely to provide such openness that IUHM has the notion of a separate descriptor and expresses thereby the relationships between data and services in terms of a hypertext network, and in this regard, IUHM supports functional, declarative programming. In contrast to the object-oriented approach, IUHM provides dynamic assignment of data to services at the level of programming in the large and supports persistent storage of contents without constraints on their actual data structure.

6.2.3 Hypertext models

IUHM is not a new hypertext model, and does not set out to be sufficiently powerful to express any hypertext domain. Rather IUHM provides an integrating structure based on a hypertext navigational paradigm in order to specify functional policy of interconnected data and services outside of these data or services. In comparison to FOHM [8], IUHM offers mechanisms to control the data as well as service border [9].

6.2.4 Annotation

Annotea [6] shares with OPALES the philosophy of defining a simple model for integrating any type of annotation. However, Annotea does not permit specification of the dynamic integration of possibly associated services: Separate applications are in charge of interpreting Annotea compliant annotation structures.

6.2.5 Web services

Web services [25] is a new philosophy of the Web to offer reusable services which must be registered, in a similar fashion to OPALES, thus offering service invocation mechanisms. In contrast to OPALES, applications are fully responsible for the definition of accessed services. There is no means of functionally defining access control.

6.3 Future Work

The success of integrating various services in OPALES suggests that our approach is promising. We intend to work on larger scale experimentation of service modelling in terms of IUHM and to consider how patterns for designing services might emerge.

7. ACKNOWLEDGMENTS

This work has been possible owing to the OPALES consortium under a PRIAMM grant from the French Ministry of Industry. We especially thank the INA and MSH teams. The work of Dr. King is supported by a research grant from the Natural Sciences and Engineering Research Council of Canada.

8. REFERENCES

[1] Anderson, K.M., Och, C., King, R., Osborne, R.M. Integrating Infrastructure: enabling large-scale client integration, Proc. ACM Conf. Hypertext'2000, ACM Press (2000), 57-66.

[2] Davis, H.C., Knight, S., Hall, W. Light hypermedia link services, a study of third party application integration, Proc. ECHT'94, ACM Press, 1994, 41-50.

[3] Earley, J. Towards an understanding of data structures, CACM, vol. 4, n° 10, Oct. 1971, 617–627.

[4] Grønbak, K. & Trigg, R. From Web to workplace: designing Open Hypermedia Systems, MIT Press, 1999.

[5] Halasz, F. & Schwartz, M. The Dexter hypertext reference model , *NIST Hypertext Standardisation Workshop*, Gaithersburg, 1990, also in *CACM*, Vol. 37 (2), (version without specification in Z), 1994, 30-39.

[6] Kahan, J., Koivunen, M.R., Prud'Hommeaux, E., & Swick R.R. Annotea: An Open RDF Infrastructure for Shared Web Annotations, in *Proc. of the WWW10 Int. Conference*, Hong Kong, (2001).

[7] Millard, D.E., Davis, H.C. Navigating spaces: the semantics of cross domain interoperability, Proc. 2nd Int. Workshop on Structural Computing, Springer-Verlag, LNCS 1903, 2000.

[8] Millard, D.E., Moreau, L., Davis, H.C., & Reich, S. FOHM: a fundamental open hypertext model for investigating interoperability between hypertext domains, Proc. Hypertext'2000, ACM Press, 2000, 93-102.

[9] Millard, D.E. Discussions at the data border: from generalized hypertext to structural computing, Journal of Network and Computer Application, Special issue on Structural Computing, Jan. 2003.

[10] Nanard M., Nanard J. Cumulating and Sharing End-Users Knowledge to Improve Video Indexing in a Video Digital Library, ACM / IEEE Joint Conf. on Digital Libraries, ACM Press, 2001.

[11] Nanard, J., & Nanard, M. Using types to incorporate knowledge in hypertext, Proc. ACM Conf. Hypertext'91, ACM Press, 1991.

[12] Nelson, H. The heart of connection: hypermedia unified by transclusion, CACM, Vol. 38, n°8, 1995, 31-33.

[13] Nürnberg, P.J., Leggett, J.J., & Schneider, E.R. As we should have thought, Proc. ACM Conf. Hypertext'97, ACM Press, 1997, 96-101.

[14] Nürnberg, P.J., Leggett, J.J., & Wiil, U.K. An agenda for open hypermedia research, Proc. ACM. Conf. Hypertext'98, ACM Press, 1998, 198-206.

[15] Osterbye, K., Wiil, U.K. The flag taxonomy of open hypermedia systems, Proc. ACM Conf. Hypertext'96, ACM Press, 1996, 129-139.

[16] Reich, S., Wiil, U.K., Nürnberg, P.J., Davis, H.C., Gronbæk, K., Anderson, K.M., Millard, D.E., & Haake, J.M. Addressing interoperability in open hypermedia: the design of the open hypermedia protocol, The New Review of Hypermedia and Multimedia, 1999, 207-248.

[17] Ross, D. T. Uniform Referents: An Essential Property for a Software Engineering Language, in J. T. Tou, ed., Software Engineering, Academic Press, 1970.

[18] Rubart, J., Wang, W., & Haake, J.M. Arguments for open execution services.

[19] Wiil, U.K., Osterbye, K. Using the Flag taxonomy to study hypermedia systems interoperability, in Proc. Hypertext'98, ACM Press, 1998, 188-197.

[20] Wiil, U.K., Hicks, D.L., & Nürnberg, P.J: Multiple open services: a new approach to service provision in open hypermedia systems, Proc. Hypertext'2001, ACM Press, 2001, 83-92:

[21] Wiil, U.K., Nürnberg, P.J. Evolving hypermedia middleware services: lessons and observations, Proc. ACM Symposium on Applied Computing, (SAC'99), ACM Press, 1999, 427-436.

[22] Wiil, U.K. Development Tools in Component-Based Structural Computing Environments, OHS7 workshop,

[23] Wiil, U.K. Toward a proposal for a standard component-based open hypermedia system storage interface, Proc. OHS6 and SC2, LNCS 1903, Springer Verlag, 2000.

[24] Wiil, U.K., Nürnberg, P.J., Hicks, D.L., Reich, S. A development environment for building component-based open hypermedia systems, Proc. ACM Conf. Hypertext'2000, ACM Press.

[25] W3C, Web services, http://www.w3.org/2002/ws/