

The HC-Data Model: A New Semantic Hypermedia Data Model

Denis Helic, Seid Maglajlic, Nick Scherbakov

Institute for Information Processing and Computer Supported New Media

Technical University Graz, Austria

Schießstattgasse 4a, 8010 Graz, Austria

Phone: ++43 316 873 5633 E-mail: dhelic@iicm.edu

Abstract: In this paper we present the current situation at the field of hypermedia data modelling. We list advantages and disadvantages of the most primitive hypermedia data model, i.e. the node-link data model and the models of the second generation hypermedia, such as the Hyperwave data model and HM-Data Model. We believe that an introduction of a semantic hypermedia data model, called the HC-Data Model can solve a number of problems in modern hypermedia data modelling. Thus, we give a formal definition of the HC-Data Model. At the end we in short present Structure Editor, an application that supports the HC-Data Model.

1. Introduction

Hypermedia combines the words "hypertext" and "multimedia".

The hypertext technology allows the user a non-linear access to information, contained in a collection of textual documents. Hypertext introduces the term of a hyperlink. A hyperlink connects two related documents. An user can follow a hyperlink and in this way jump from the document he/she currently reads to a related document [5].

While hypertext uses only textual documents to present information, multimedia offers a possibility to use other forms of information. One may incorporate graphics, pictures, sounds, animation, video and combine them to a single multimedia document [6].

Hypermedia combines these two technologies into one new. It is based on the hypertext technology, in the sense that it connects documents via associative links, but in the opposite to hypertext, each document is a multimedia document and can have a number of different media objects [5].

Hypermedia technology is a special technology that deals with a big number of collections of multimedia documents. More precisely, hypermedia deals with data structures imposed on the top of such collections. Thus, we can speak about hypermedia as a special kind of database. A hypermedia data model determines data structures used in a particular system.

There are several different hypermedia data models today. Currently, the most popular hypermedia data model is the node-link data model and is being widely applied. But the node-link data model is also the most primitive one. In this model, a particular multimedia document is seen as a node and relations between nodes are denoted as links. Links are always directed, i.e. there is a source document and there is a destination document. Links are embedded into nodes, that is each link is a part of the document definition and is stored in the database as such.

There is a number of problems connected to the use of the node-link data model:

- editing of links is tedious
- logical integrity is not supported
- links are not context-dependent
- the presence of links unrelated to the current context leads to reader disorientation
- the reuse of hypermedia materials is unsatisfactory

Hypermedia systems can be roughly classified into two groups:

- stand-alone hypermedia systems offering access to documents stored on a single machine
- distributed hypermedia systems offering access to documents stored on different machines in a network

Widely, the most used distributed hypermedia system is WWW. WWW is based on three main concepts:

- URL
- HTTP
- HTML

URL (Uniform Resource Locator) is an unique address of a hypermedia document on the Internet. It holds information about the host machine where the document resides, and about the position of the document on the host machine.

HTTP (HyperText Transfer Protocol) is an Internet protocol that allows a WWW client (WWW browsers) to request a hypermedia document from a WWW server using their URLs. Documents are then transferred from the server to the client in the form of a HTML document and are rendered into multimedia information inside of a WWW browser.

HTML (HyperText Mark-up Language) is a mark-up language, used to store hypermedia information on the WWW server side, as well as to give instruction to the WWW client how to render the document on the user screen. It uses different mark-up tags to embed different types of media objects, such as pictures, video, animation, etc. Further, it allows embedding of URLs of another documents used then as hyperlinks. Thus, HTML implements node-link data model and inherits in this way its disadvantages.

Previous discussion shows that even HTML is de facto standard for visualising hypermedia documents, an usage of HTML as a data structuring paradigm for hypermedia database has a number of disadvantages. All these disadvantages have their roots in the primitive node-link data model, which is supported by HTML. Thus, an use of another logical data model, that offers different structuring mechanism to hundreds of hypermedia documents and that solves problems of the node-link data model must be seriously considered.

The most important requirement on a new logical data model should be the separation of the structure of data from its actual content as it is the case in traditional databases. Furthermore, the integrity of hypermedia database should be guaranteed, as well as the link consistency as a specific property of hypermedia system. And last but not least the hypermedia material must be easily reusable and if needed changed by an author and adjusted to his specific needs. Therefore, the term of hypermedia composite was introduced. The hypermedia composite presents a closed collection of hypermedia documents that has some internal link structure, which may be created automatically and which ensures the integrity of the database.

These hypermedia data models can be seen as hypermedia data models of second generation and two of them: the Hyperwave [1] and the HM-Data Model [6] are already introduced. Nevertheless, an usage of new logical data models has also its disadvantages, such as:

- there is a need to learn how to use a new logical data model (this might last a long time)
- it is difficult to use a logical data model in a big number of different applications (most of them are adjusted to some specific type of hypermedia applications).

Thus, as we can see, the introduction of new logical data models which solve the problems rising from the lack of highly structured data constructs in the node-link data model, i.e. the database integrity, the link consistency, the satisfactory reuse of material, brings a number of new problems [2]. We believe that the use of a new semantic hypermedia data model called the HC-Data Model, that gives a meaning to a particular hypermedia composite and its elements could solve these new problems. Because of self-explanatory nature of each semantic hypermedia composite the learning time is much shorter, if there is one. On the other hand, the HC-Data Model provides a facility for defining new types of hypermedia composites, that fits into demands of a particular hypermedia application.

Following this leading motive, the HC-Data Model with its great flexibility could be used in a number of very different applications [3].

2. HC-Data Model

HC-Data Model is a semantic hypermedia data model that operates on HC-Units and HC-Types. The prefix HC stands here for Hypermedia Composite. Like semantic data models in traditional databases a semantic hypermedia data model models the hypermedia information as an abstract entity with a number of specific characteristics. While a logical data model describes how the information is structured, a semantic data model handles abstract data entities that have certain properties. For example a classical semantic data model would model a person as an abstract data object **Person** that has a *name*, an *age* and a *profession* as its properties. Similarly a semantic hypermedia data model would comprehend a person as an abstract data object **Person** that has hypermedia/multimedia documents containing a *biography*, a *picture* and a description of *current work* as its properties. A semantic data model works on instances of abstract data types defined in the model. An abstract entity **Person** that has a *name*, an *age* and a *profession* can be considered to be an abstract data type, and a *person* with the *name Nick*, the *age 45* and the *profession Scientist* can be seen as an instance of abstract data type **Person**. In the HC-Data Model an HC-Type is an abstract data type definition and an HC-Unit is an instance of an HC-Type.

In traditional data modelling a semantic data model was first step from older logical data models, such as relational data model, to object-oriented data models. In contrast with object-oriented data models a semantic data model does not describe the behaviour of an abstract data type, but only its characteristics. However, the development of hypermedia data models differs greatly from the development of traditional data models. After many of disadvantages of the most primitive hypermedia data model, the node-link data model, were recognised, the second generation hypermedia was introduced. Data models, such as Hyperwave data model or HM-Data Model presented the concept of hypermedia composite, which was already "seriously" object-oriented. A hypermedia composite was defined as a hypermedia object that is a collection of hypermedia documents and/or other hypermedia composites. Furthermore, both of mentioned hypermedia data models defined the public interface to manipulate the private memory of their hypermedia composites, i.e. to insert, add, replace, delete members of hypermedia composites or to access and browse hypermedia composites. The Hyperwave data model, likewise the HM-Data Model had a number of predefined types of hypermedia composites that was hardly extendable. The HC-Data Model extends the object-oriented concept of the hypermedia composite with the possibility to define new types of hypermedia composites. This is done via the concept of an HC-Type which is an abstract data type definition or we can say a meta-definition of a class of hypermedia composites. Instances of an HC-Type are called HC-Units of this type. Compared with a traditional semantic data model, which was the preceding step to the introduction of object-oriented data models, the HC-Data Model, as a hypermedia semantic data model, is a step which follows after the introduction of object-oriented hypermedia data

models. The HC-Data Model sees a hypermedia composite as an abstract data entity that has a number of characteristics. These characteristics are determined by an abstract data type definition, i.e. by an HC-Type.

An HC-Type defines characteristics of hypermedia composite and its members, that is it defines a number of meta-information attributes which will be attached to instances of that HC-Type, i.e. HC-Units and their members. The meta-information attributes appear in the form of key-value pairs. Some of these attributes are fully defined, that is to say both, the key and the value, are specified in an HC-Type. For some of these meta-information attributes only the name of the key is defined whereas the value will be entered when manipulating an HC-Unit of that HC-Type. We can classify those meta-information attributes into two groups:

- meta-information attributes of a hypermedia composite
- meta-information attributes of members of a hypermedia composite

A correct definition of a hypermedia composite type, that is a correct HC-Type has to define following attributes from the first group:

- CompositeType attribute with an assigned value
- CompositeAccessView attribute with an assigned value
- CompositeURL attribute without a value.

From the second group an HC-Type has to include the definition of at least one member of that HC-Type (but can and probably will have the definition of a number of different members). This definition includes following meta-information attributes:

- MemberRole attribute with an assigned value
- MemberPosition attribute without a value
- MemberURL attribute without a value.

Additional attributes can also be defined for both hypermedia composite and/or its members with or without assigned values.

For example an abstract data object **Person** that we already mentioned, defined as an HC-Type, would have following attributes:

- CompositeType attribute with the value *Person*
- CompositeAccessView attribute with the value *file://c:/composites/person.view*
- CompositeURL attribute without a value

The "Person" HC-Type would have three different members with following attributes:

- First Member:
 - MemberRole attribute with the value *Biography*
 - MemberPosition attribute without a value
 - MemberURL attribute without a value
- Second member:
 - MemberRole attribute with the value *Picture*
 - MemberPosition attribute without a value
 - MemberURL attribute without a value

- Third member:
- MemberRole attribute with the value *CurrentWork*
- MemberPosition attribute without a value
- MemberURL attribute without a value

An HC-Unit of the **Person** HC-Type, that is an instance of the **Person** class of hypermedia composites might look as follows:

- CompositeType attribute with the value *Person*
- CompositeAccessView attribute with the value *file://c:/composites/person.view*
- CompositeURL attribute with the value *file://c:/scratch/nick.htm*

The members could be:

- First member:
- MemberRole attribute with the value *Biography*
- MemberPosition attribute with the value *1*
- MemberURL attribute with the value *file://c:/nick/biography.htm*
- Second member:
- MemberRole attribute with the value *Picture*
- MemberPosition attribute with the value *1*
- MemberURL attribute with the value *file://c:/nick/nick.gif*
- Third member:
- MemberRole attribute with the value *CurrentWork*
- MemberPosition attribute with the value *1*
- MemberURL attribute with the value *http://www.iicm.edu/nick/work.htm*

Furthermore, an HC-Type offers a public interface to manipulate the content of an instance private memory. This public interface includes following methods:

- *public void add(String member_role, URL url);*
- *public void replace(String member_role, URL url);*
- *public void insert(String member_role, int member_position, URL url);*
- *public void delete(String member_role, int member_position);*
- *public CompositeAccessView access();*

The *add()* method adds a member, which might be a hypermedia document or an HC-Unit, to an HC-Unit giving it the specified MemberRole attribute. This method is used whenever more than one member can have the same MemberRole attribute. In this case our HC-Unit has a list of members with the same MemberRole attribute but the different values of the MemberPosition attribute. The value of the MemberPosition attribute is equal to the position of a member in the list. Whenever the *add()* method is used the new member is added at the end of this list so the value of its MemberPosition attribute is set to the current size of the list incremented by one.

The *replace()* method is used when a member should get the specified *MemberRole* attribute but this attribute can be assigned exactly to the one member. Then the old member is replaced by the new one. The value of the *MemberPosition* attribute of such a member is always equal to 1.

The *insert()* method is used when a member has to be inserted in a list of members having the specified *MemberRole* attribute exactly on the specified position. In this way the value of the *MemberPosition* attribute of the new member is set to the specified position and the values of the *MemberPosition* attribute of members following the inserted one in the list is incremented by one.

The URL *url* argument of these three methods specifies an address of a hypermedia document or an HC-Unit on the Internet or a local file system. It is used by the access method in order to visualise an HC-Unit properly.

The *delete()* method deletes specified member from an HC-Unit. If the deleted member is in a list of a *MemberRole* attribute then the values of the *MemberPosition* attribute of other members in this list will be updated.

The *access()* method is used in order to visualise an HC-Unit following to the definition given in a special file (this file is specified through the "CompositeAccessView" attribute of an HC-Type) by using the existing hypermedia documents and/or existing HC-Units located by the "MemberURL" attributes.

3. Visualisation of an HC-Type

An HC-Type provides the *access()* method in its public interface. The *access()* method is called whenever an HC-Unit of that HC-Type is accessed in the sense of browsing the hypermedia information that the accessed HC-Unit contains. The CompositeAccessView attribute of an HC-Type has as its value an URL of a file defining the visualisation mechanism for the particular HC-Type. When the *access()* method of an HC-Unit is called for the first time the definition file is interpreted and a number of visualisation objects is instantiated. These visualisation objects became the part of that HC-Unit private memory and are responsible for the proper visualisation of the particular HC-Unit. They provide also an user interface for retrieving the hypermedia information, i.e. for browsing the content of an HC-Unit.

The most important visualisation object is called **ScreenTemplate**. A **ScreenTemplate** is a rectangular area representing a canvas for visualising an HC-Unit. This canvas is split into a number of rectangles which do not intersect with each other and whose union is equal to the starting rectangular area of the particular screen template. We will call these rectangles screen template cells or just cells. Each screen template cell has a so-called **PlaceHolder** visualisation object as its child. A **PlaceHolder** object is an object that presents a piece of information from a particular HC-Unit. The rectangular area of the screen template cell which is the parent of the particular **PlaceHolder** object, is filled with information from the **PlaceHolder** object. As a result of filling of all screen template cells with the corresponding piece of information we obtain a filled **ScreenTemplate** object which is presented to an user. Because an HC-Unit is comprised of hypermedia information a **ScreenTemplate** object is composed as a valid HTML document and consequently rendered in a standard Web browser.

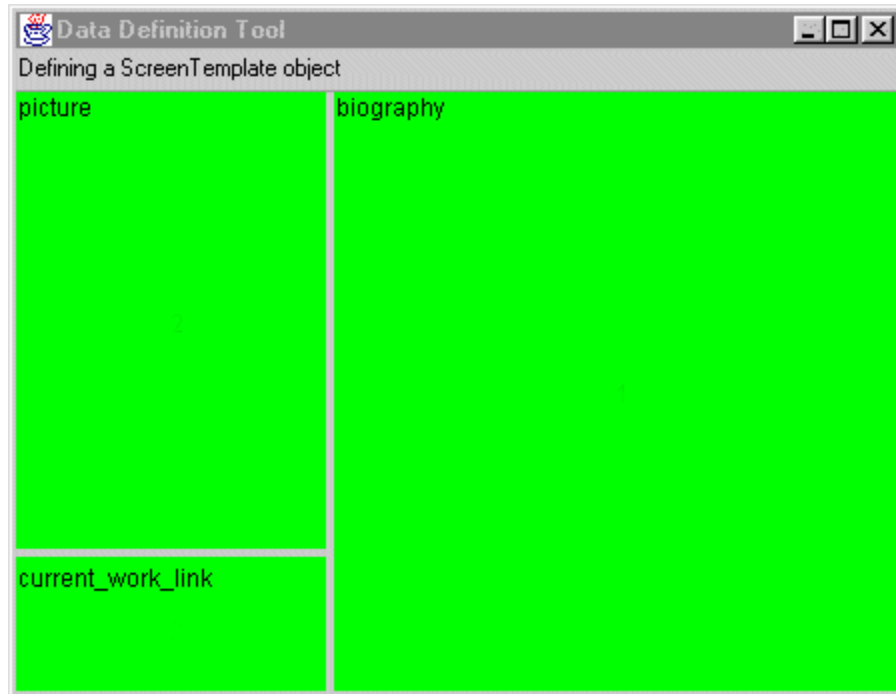


Fig. 1. Defining a ScreenTemplate object with a number of Placeholder objects

An HC-Type can and probably will define a number of **ScreenTemplate** objects for visualisation of HC-Units of that type. One of these **ScreenTemplate** objects is defined as a starting one, i.e. it will be shown whenever the *access()* method of the particular HC-Unit is called. An user will activate computer-navigable links in order to access other **ScreenTemplate** objects. As a result of accessing another **ScreenTemplate** object the starting one will be removed and the content of the accessed one will be rendered. Defining a navigable structure of an HC-Type is done via defining **Placeholder** objects with special linking property, that is to say via defining **Placeholder** objects which are computer-navigable links.

Let us now classify **Placeholder** objects. There are three different groups of **Placeholder** objects:

- **Placeholder** objects with a fixed content
- **Placeholder** objects with a dynamic content
- **Placeholder** objects representing computer-navigable links

Placeholder objects with a fixed content are the **TextPlaceholder** object and the **ImagePlaceholder** object.

The **TextPlaceholder** object has as the information source a fix defined piece of plain or HTML text. This fixed text that a **TextPlaceholder** object holds is presented in a screen template cell of this object.

Similarly, a **ImagePlaceholder** object has as its content a fixed picture which is shown in the screen template cell that this object occupies.

Placeholder objects with a dynamic content can be the **MemberPlaceholder** object or the **AttributePlaceholder** object.

A **MemberPlaceholder** object has as the information source a member of an HC-Unit. The member is strictly defined through the *MemberRole* meta-information attribute, by which is meant that a

MemberPlaceHolder object holds the place for a member with the specified value of the *MemberRole* attribute. For instance in the HC-Type **Person** we will have a **MemberPlaceHolder** that holds the place for a member that has *Biography* as the value of the *MemberRole* attribute. Using the *MemberURL* attribute of a member the actual content of this member will be rendered in the screen template cell of the particular **MemberPlaceHolder**. If a number of members has the same *MemberRole* attribute, that is if an HC-Unit has a list of members of a particular *MemberRole* attribute then the value of the *MemberPosition* attribute should be specified in a **MemberPlaceHolder** object in order to determine the right member. The value of the *MemberPosition* attribute in a **MemberPlaceHolder** may be altered, that is it can be incremented or decremented. In other words an user can navigate through the list of members having the same value of a *MemberRole* attribute. This navigation occurs also as a result of following computer-navigable links defined through a special kind of **PlaceHolder** objects representing hyperlinks.

In the same way as a **MemberPlaceHolder** object a **AttributePlaceHolder** object uses a value of a *MemberRole* and a *MemberPosition* attribute to point out a member of an HC-Unit, but while a **MemberPlaceHolder** object uses the *MemberURL* attribute of the particular member a **AttributePlaceHolder** uses a name of some other attribute of the particular member in order to display the value of this attribute. If the *MemberRole* attribute of a **AttributePlaceHolder** object has a list of members **PlaceHolder** objects representing computer-navigable links can be used in order to navigate through the member list.

PlaceHolder objects representing computer-navigable links are not a special group of **PlaceHolder** objects but a **TextPlaceHolder** object, a **ImagePlaceHolder** object or a **AttributePlaceHolder** object can be defined to be also a hyperlink. We will call these objects a **LinkPlaceHolder** object. For example the functionality of a **TextPlaceHolder** object can be extended to the functionality of a **LinkPlaceHolder** object of the **TextPlaceHolder** object type, by which is meant that the text of a **TextPlaceHolder** has been put inside of an HREF tag. The destination document of a **LinkPlaceHolder** object can be another **ScreenTemplate** object of that HC-Type or the same **ScreenTemplate** object with altered values of the *MemberPosition* attributes of **MemberPlaceHolder** or **AttributePlaceHolder** objects. In this way browsing of an HC-Unit is carried on.

For instance our HC-Type **Person** can be defined to have two **ScreenTemplate** objects:

- **ScreenTemplate** object called *access* defined as the starting one
- **ScreenTemplate** object called *current_work*

The *access* **ScreenTemplate** object has three **PlaceHolders** as follows:

- **MemberPlaceHolder** object called *biography* that holds a member with the *MemberRole* attribute of the value *Biography*
- **MemberPlaceHolder** object called *picture* that holds a member with the *MemberRole* attribute of the value *Picture*
- **TextPlaceHolder** object called *current_work_link* that holds a fixed text (e.g. See my current work) and that is defined as a **LinkPlaceHolder** object which replaces the *access* **ScreenTemplate** object with the *current_work* **ScreenTemplate** object.

The *current_work* **ScreenTemplate** object has two **PlaceHolders** as follows:

- **MemberPlaceHolder** object called *current_work_member* that holds a member with the *MemberRole* attribute of the value *CurrentWork*
- **TextPlaceHolder** object called *go_back_link* that holds a fixed text (e.g. Go back) and that is defined as a **LinkPlaceHolder** object which replaces the *current_work* **ScreenTemplate** object with the *access* **ScreenTemplate** object.

4. Implementation of HC-Data Model

Structure Editor is a system that supports and implements the HC-Data Model. It comprises three components:

- Structure Editor Visual Definition Tool
- Structure Editor Visual Manipulation Tool
- Structure Editor Java Visualisation Interpreter

With the help of Structure Editor Visual Definition Tool a new HC-Type can be defined. The tool offers a graphic user interface with drag and drop facilities. An author firstly defines a new abstract entity through specifying its meta-information attributes and finally he/she defines a visualisation mechanism via defining a number of ScreenTemplate objects with corresponding Placeholder objects.

The newly defined type, HC-Type, is then used by Structure Editor Visual Manipulation Tool in order to create instances of the HC-Type, i.e. HC-Units – collections of existing hypermedia documents and/or other HC-Units. This tool offers also an easy-to-use graphic user interface again with drag and drop facilities. Wanted hypermedia documents are then simply dragged from a tree-like structure, which can be browsed in the sense of Windows Explorer, into a right place in an HC-Unit. Hypermedia information can be accessed from any kind of information source, such as a Hyperwave information server, an HTTP server, an FTP server or a local file system. This is done through so-called ANT (Active Node Technology) [4] which allows a fully transparent access to any kind of information stored on any type of information server (see fig. 2.).

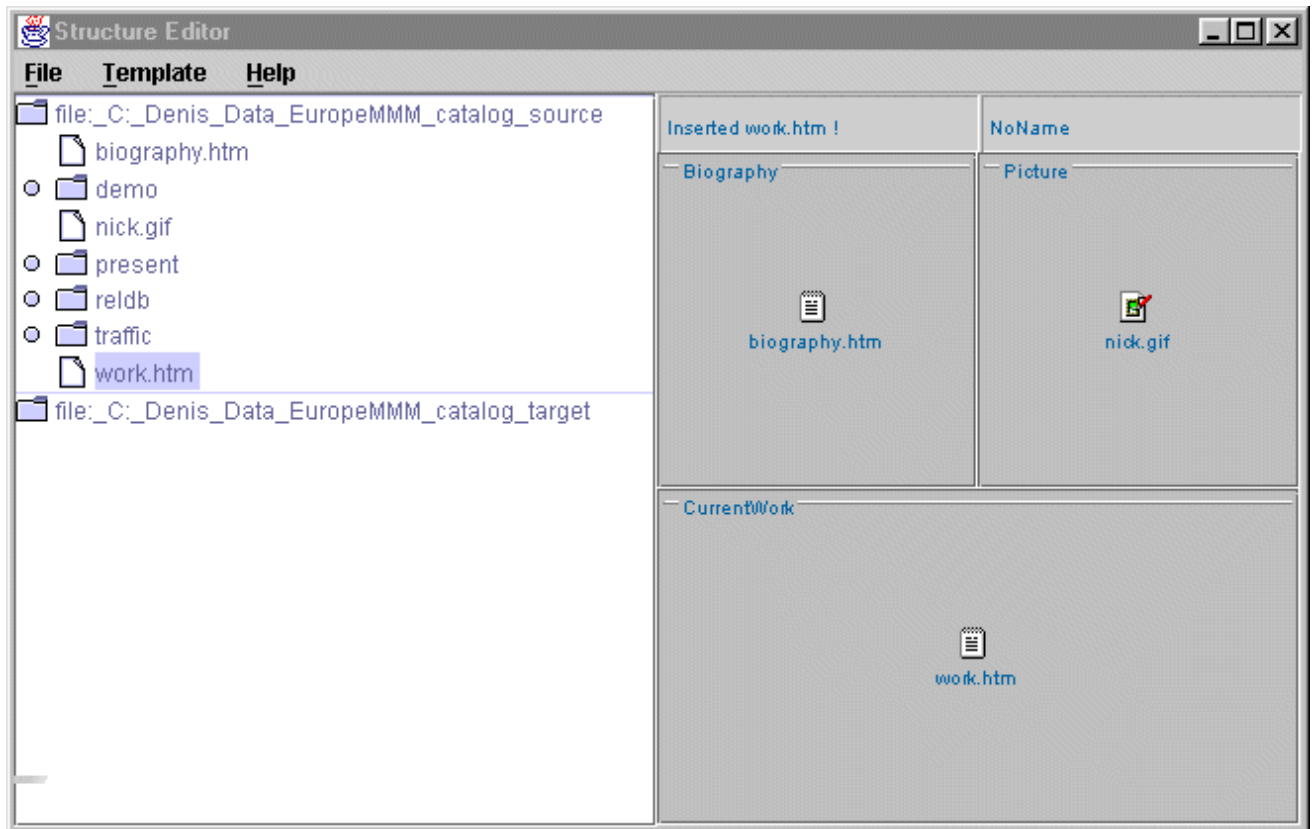


Fig. 2. Structure Editor Visual Data Manipulation Tool



Fig. 3. Accessing an instance of the Person HC-Type

Once an HC-Unit has been created it can be published on a Web information server or a local file system. It can be then accessed by any standard Web browser, such as Netscape Navigator or Microsoft Internet Explorer. The visualisation is performed by the definition of corresponding HC-Type and controlled by the third Structure Editor component, i.e. Structure Editor Java Visualisation Interpreter (see Fig. 3.). This interpreter is implemented in the form of a Java applet and is embedded in the starting page of a created HC-Unit.

5. Conclusion

In conclusion, we would like to notice that the proposed HC-Data Model as a semantic hypermedia model that extends the possibilities of the object-oriented hypermedia data models, such as the Hyperwave data model or the HM-Data Model, can have even bigger possibilities if used in a combination with some of these models. Mapping of the HC-Data Model to the Hyperwave data model is a good example of this combined use and is described in [3]. Even mapping of the HC-Data Model to the node-link data model is possible and needed because of a big number of standard HTTP servers installed world-wide. But the most interesting application of the HC-Data Model would be, as we truly believe, mapping of the HC-Data Model onto XML. Similarly to the HC-Data Model XML technology [7, 8] allows an author to define new types of hypermedia constructs, i.e. the structure of hypermedia database, as well as the navigational and visualisation paradigm. Because XML becomes already a standard the combination of the HC-Data Model and XML technology can offer the best possibilities for creating highly structured hypermedia applications.

References

1. Hermann Maurer, "Hyperwave – The Next Generation Web Solution", *Addison-Wesley*
2. Denis Helic, Hermann Maurer, Nick Sherbakov, "Authoring and Maintaining of Educational Applications on the Web", *EDMEDIA 99*
3. Denis Helic, Seid Maglajlic, Nick Sherbakov, "Educational Materials on the Web: Data Modelling Approach", *MIPRO 99*
4. D. Freismuth, K. Schmaranz, B. Zwantschko, "Telematic Platform for Patient Oriented Services", *JUCS, vol. 11, 1998*
5. H. Maurer, N. Scherbakov, Z. Halim, Z. Razak, "From Databases to Hypermedia", *Springer*
6. H. Maurer, N. Scherbakov, "Multimedia Authoring for Presentation and Education", *Addison-Wesley*
7. Microsoft Online XML Workshop, <http://www.microsoft.com/xml>, *Microsoft Corp. Homepage*
8. Extensible Markup Language (XMLTM), <http://www.w3.org/XML/>, *World Wide Web Consortium*