Browsing Hypermedia Composites: An Algebraic Approach

Hermann Maurer, Keith Andrews and Nick Scherbakov
Institute for Information Processing and Computer Supported New Media,
Graz University of Technology, Schießstattgasse 4a,
A-8010 Austria.
Tel.: 43/316/83-25-51/Ext. 12 Fax: 43/316/82-43-94
Email: nsherbak@iicm.tu-graz.ac.at

Abstract: This paper presents an algebraic approach to the browsing of hypermedia composites, based on a fully object-oriented, logical data model for hypermedia called the HM Data Model.
The main unit of abstraction in the HM Data Model is an object called an S-collection. Hypermedia links are encapsulated within a particular S-collection and are bound between S-collections. The algebraic operations INCLUDE, JOIN and DIVIDE can be applied to S-collections to temporarily create new views of the hypermedia structure.

1. Introduction

Hypermedia data models serve as necessary abstractions in order to combine large amounts of raw multimedia information into a logically integrated hypermedia database and provide users with a well-defined set of operations for creating, modifying and accessing such a database.
In fact, the term hypermedia was born from a particular data model called the *node-link* paradigm. In this basic hypermedia paradigm, information is stored as a so-called hyperweb (i.e. as a collection of primitive *nodes* , with *links* between nodes representing different relationships). In this node-link data model, access to information is achieved by freehand browsing of the hyperweb. The *default browsing semantics* introduced by the basic node-link paradigm and then adopted by nearly every logical hypermedia data models can be defined as follows: if there exist a number of computer-navigable links emanating from a particular node, then any link can be manually selected in order to traverse the hyperweb to the destination of this link (i.e. in order to access another node) where the process is repeated.
As hypermedia systems become larger, more dynamic and possibly distributed, it becomes less and less possible to create and maintain the entire hyperweb of information solely by means of the basic node-link hypermedia paradigm
In this paper, we discuss a so-called HM Data Model and offer algebraic operations to extend the navigational component. The rest of the paper is structured as follows: Section 2 contains a brief description of the HM Data Model and an illustrative example. In Section 3 we discuss the algebraic approach to browsing hypermedia composites supported by the HM Data Model. The paper ends with some concluding remarks.

2. The HM Data Model

2.1. Data Structure Types

The HM Data Model defines a particular hypermedia database as a set of abstract data objects called *S-collections* ("structured-collections"). S-collections are created as instances of previously defined system classes.

Every S-collection (instance) has a unique identifier, a name used by other S-collections for addressing, and generally a "data unit" which will be called *content* henceforth. Executing the content will ordinarily cause some information to be presented (text, pictures, audio, video-clips, etc.).
An S-collection encapsulates a particular internal structure. The internal structure is a set of other S-collections (called *members* henceforth) related by a number of computer-navigable links. One of the members of the S-collection must be designated as *head* of the S-collection. Note that links are encapsulated within a particular S-collection: they may only be defined between members of the same S-collection. In this sense, links belong to a particular S-collection; they do *not* belong to the hypermedia database or to either of the members related by the link (hence the "local referential integrity" [11] of the model). Links also do *not* exist as independent data objects which are instances of a particular class, operations cannot be addressed to links.

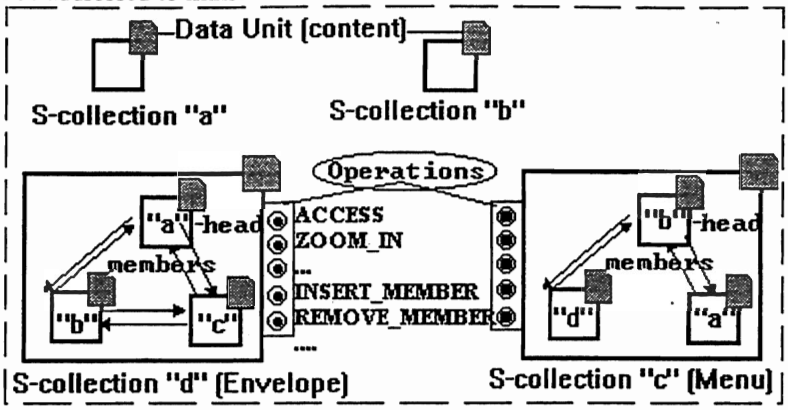

Figure 1: Internal Structure of S-collections

The HM Data Model provides a number of predefined subclasses of S-collection. From a data structuring point of view, these predefined subclasses are similar to the hypermedia topologies introduced by Parunak [18]. Simply speaking, subclasses of S-collection define a particular topology of encapsulated links, as can be seen in Figure 1. For the purposes of this paper, it suffices to describe the five predefined subclasses : Envelope, Folder, Menu, Freelinks, and Void:
1. **Envelope**: all members of an envelope are fully related, every member is linked to every other member.
2. **Folder**: an ordered set of members, each member having links to "next" and "previous" members.
3. **Menu**: a simple hierarchical structure; the head of a menu S-collection includes links to all other members, and each member is provided with a link to the head.
4. **Freelinks**: members of a freelinks S-collection may be arbitrarily connected by means of special INSERT_LINK and REMOVE_LINK operations.
5. **Void**: an S-collection devoid of internal structure, having only content (for example, "a" and "b" in Figure 1).
Classes 1 to 4 above are termed *complex* S-collections since they embody internal structure; class Void is termed *simple*. Complex S-collections may or may not have associated content; simple S-collections must have associated content.

2.2. Navigational Operations

Navigation within the HM Data Model is accomplished via four operations ACCESS, ZOOM_IN, ZOOM_UP and ZOOM_BACK, which are addressed to a particular S-collection.
All S-collections can respond to the message ACCESS. It implies executing the S-collection's content (i.e., presenting some text, picture, audio, video clips etc.). Typically, a chunk of hypermedia information associated with the current S-collection is visualised on the user screen, but any kind of action can happen in response to the message ACCESS if the corresponding method has been overridden. Complex S-collections without content forward the message to their head.

Link following within the HM Data Model is simply a form of message passing. At any particular moment in time, the user can navigate only through a single, specific S-collection called the *current container*. Only members of the current container can receive messages during navigation. A concrete member of the current container is the *current member* for each particular navigational step. More precisely, the member that most recently received the message ACCESS is the current member. Only members related (linked) to the current member can be accessed (can receive the message ACCESS) in the next step of navigation.
Consider, for instance, navigation through S-collection "d" in Figure 1. S-collection "d" is the current container. If S-collection "a" is the current member, then it has been visualised (its content has been displayed by method ACCESS), and the user has links to members "b", and "c" available in the next step of navigation. In our prototype implementation, links emanating from the current member are depicted in the form of icons, buttons, clickable areas, etc. The user manually selects a link which results in the message ACCESS being sent to the corresponding member.
Since links are encapsulated within an S-collection, they become available for navigation only when the S-collection has been "entered" by means of the ZOOM_IN operation, which is available for all complex S-collections. The ZOOM_IN message is automatically addressed to the current member. For example, if the S-collection "c" becomes a current member during browsing current container "d" (see Figure 1), the user can apply the ZOOM_IN operation in order to make "c" the new current container. After a ZOOM_IN operation, the head of the new current container (i.e. S-collection "b" in this particular case) automatically becomes the current member and is visualised appropriately (i.e. receives message ACCESS).
Extending the functionality of ZOOM_IN to give access to any S-collection of which the current collection is a member is provided by the operation ZOOM_UP. Thus, if current member (say, S-collection "a") has been accessed during browsing the current container "d", then the user can "switch" to browsing of any other current container having the same S-collection "a" as a member (say, to browsing the container "c" in this particular case). Thus, the ZOOM_UP operation simply substitues the current container for another one leaving the current member unaffected.
The ZOOM_BACK operation is the complement of ZOOM_IN or ZOOM_UP. ZOOM_BACK restores the current container and current member to the state they had before the *most recent* ZOOM_IN or ZOOM_UP.
Together, ZOOM_IN, ZOOM_UP and ZOOM_BACK provide users with the capability of navigating in a direction orthogonal to the conventional plane of link-based browsing. Thus, we say that the HM Data Model supports an additional dimension of browsing hypermedia databases and extends the default browsing semantics.

2.3. Operations for Modifying the Database

New S-collection instances are created with the operation CREATE, whereby the name, content, and head are given as parameters. Once an S-collection has been created, new members can be inserted and existing members removed by means of the operations INSERT_MEMBER and REMOVE_MEMBER having a name of another S-collection (i.e. member) as a parameter. All links within S-collections belonging to the classes Menu, Envelope, and Folder are maintained automatically in accordance with their associated regular structure.
Of course, the regularly structured subclasses do not restrict us from using S-collections having arbitrarily connected members. Users can create an S-collection of class Freelinks and explicitly define its link structure using the messages INSERT_LINK and REMOVE_LINK whereby the source and destination are given as parameters.
The notion of links presented here should not be confused with that used in other hypermedia models, where links exist as independent fully-fledged persistent data objects. In our case, links and membership are simply mechanisms for reusing abstract data objects in different contexts via a special public interface. Notice that we do not map S-collections onto a global navigational level consisting of primitive nodes and links [1, 2, 21], rather we treat them as existing accessible chunks of hypermedia information.
All simple S-collections (those having content but devoid of internal structure) belong to class Void, where the operations ZOOM_IN, INSERT_MEMBER, and REMOVE_MEMBER are suppressed.
At any point, an S-collection can be deleted with the DELETE operation. In this case, its content and all links encapsulated within it cease to exist. If a deleted S-collection has been reused (defined as a member) by

other S-collections it is removed from S-collections by means of the REMOVE_MEMBER operation (see Figure 1).
Thus, the traditional linking of chunks of multimedia data used by nearly all contemporary hypermedia data models, is replaced with the insertion/removal of S-collections into/from the private memory of other S-collections. Note also that in the HM Data Model, membership is not restricted to a hierarchical structure, recursive membership is both possible and meaningful.

2.4. Illustrative Example

As an example of the application of the HM Data Model, consider the situation in Figure 2. Suppose that we would like to develop a hypermedia presentation of our institute, the IICM. A typical hypermedia database according to the HM Data Model might represent the institute as a whole as the S-collection "IICM" in Figure 2.
The content (data unit) of the S-collection "IICM" might typically include the institute logo and a short textual description of the collection (e.g. "Welcome to IICM. Click now on Zoom In in order to get started"). Suppose that "IICM" is an S-collection of type Menu and has other members "Projects", "Personnel", etc.
If "IICM" is the current container and the member "Personnel" which is an S-collection of type Folder, is accessed (clicked), its content (e.g. the text "This collection consists of personal files of IICM employees. To look through them one-by-one click now on Zoom In") is visualised. If the user now clicks on button "Zoom In", the first file (i.e. the content of S-collection "Hermann") is shown; the remaining files can be traversed by clicking buttons "next" or "previous" (the navigational aids available in a folder). The operation ZOOM_UP provides access to any S-collections of which the person is a member (say, to the S-collection "Hyper-G" in this particular case). Clicking on button "Zoom Back" returns the user to S-collection "IICM" and its navigational paradigm. Now clicking on another member of "IICM" (say, on "Projects") visualises that member's content, clicking on button "Zoom Back" leaves "IICM", and so forth.

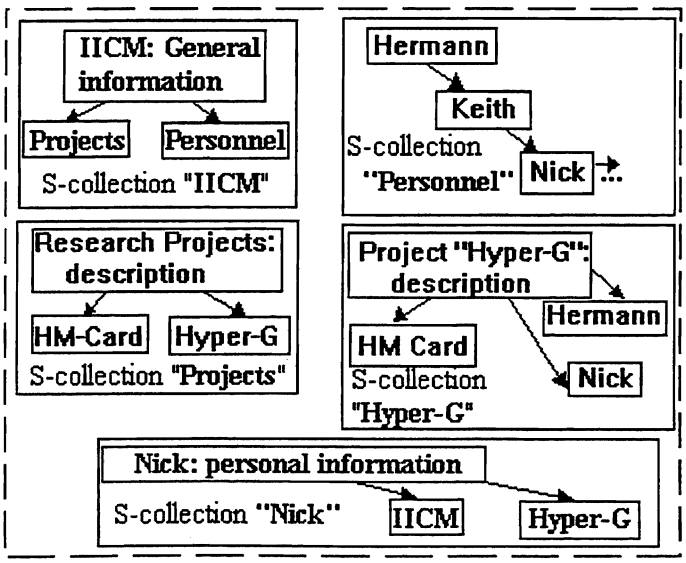

Figure 2: Example of a Database Structured Using S-collections

This situation is typical of the navigational process in the HM Data Model and should be clearly understood: when a user accesses an S-collection *B* as a member of S-collection *A*, the navigational paradigm of *A* continues to be available. Only when *B* is entered (operation ZOOM_IN), is the navigational paradigm of *B* activated, and the navigational paradigm of *A* becomes unavailable. When now selecting a member *C* of *B*, *B* and *C* start to play the roles of *A* and *B*, respectively. Of course, the ZOOM_BACK operation allows backtracking to the navigational paradigm of *A*.

A hypermedia description of a particular person (say "Nick") can be enhanced with S-collections describing projects this particular person is involved with and an S-collection describing his/her affiliation (say, "Hyper-G" and "IICM" in this particular case). Further, information about employees engaged in a particular project can be inserted into a corresponding S-collection as members (say, "Hermann" and "Nick" in this particular case). Of course, all S-collections describing different projects can be combined into an S-collection "Projects".
The property of the HM Data Model that an S-collection may belong to many other S-collections, even recursively, provides all the necessary power to deal with more complex situations. Thus, for instance, if it is desirable to have references to descriptions of all projects sharing particular equipment within the description of this equipment, the corresponding "Project" S-collections are simply inserted into the S-collection "Equipment". Note the recursive membership of S-collections "Hyper-G" and "Nick" in this example. Such recursive membership elegantly handles the common situation where a user needs to access information about "Hyper-G" while browsing all information related to "Nick", or vice versa. Moreover, if a certain project (say, "Hyper-G") is based on other projects (say, "HM-Card"), then the description of the S-collection "Hyper-G" can be extended with the relevant members (with the S-collection "HM-Card" in this particular case).
Note that the concept of encapsulation of links makes this process extremely flexible. Any S-collection can be inserted into any other S-collection (message INSERT_MEMBER) without considering the links directed to or emanating from it: new links having well-defined context (that of the containing S-collection) are added automatically; additional links can be added manually if desired (method INSERT_LINK which can be inherited from the system class Freelinks). The asymmetry of most hypermedia models (some objects belong to one part of the database and other parts just link to it) disappears: an abstract object belongs in exactly the same way to all containers, wherever it is relevant.

3. Algebraic Operators

Algebraic operators are applied to S-collections and produce a new resultant S-collection which becomes the current container. Thus, these operators provide users with the possibility to adjust the current scope of link-based navigation to their particular *short-term* needs. The algebraic operators broaden the concept of current container (see Section 2.2) and serve as an additional tool for browsing hypermedia composites.
The operator INCLUDE is applied to the current container and one of its members; the operator temporarily replaces the member with its internal navigable structure. Thus, for instance, if the user INCLUDEs the members "Projects" and "Personnel" during browsing current container "IICM" (see Figure 2), the scope of link-based browsing is extended; and the resultant current container is shown on Figure 3a.
The following simple rules are used in order to avoid possible ambiguity:

1. Links emanating from (or directed to) a member which has been replaced with its internal structure, emanate from (or lead to) its head (see Figure 3a);
2. It is possible for an S-collection to be both a member of the current container and the included S-collection. In this case, the S-collection appears only once (is not duplicated) with both sets of links. For instance, a current container which is a result of INCLUDing the S-collection "Hyper-G" in the current container "IICM" (see Figure 3a) is shown on Figure 3b.

Thus, the current container has been composed from the navigable structures of a set of S-collections. Such set of S-collections is called a *perspective*. For instance, the perspective for the current container shown on Figure 3b, consists of the S-collections "IICM", "Personnel", "Projects" and "Hyper-G".,
The operator JOIN is applied to a current container and another S-collection sharing a common member, the operator simply extends the current container with the internal navigable structure of the second S-collection. Thus, for instance, if the user JOINs the current container "Projects" (see Figure 2) and the S-collection "Nick" sharing a common member "Hyper-G", the scope of link-based browsing is extended; and the user continues navigation in the current container shown on Figure 4.
The operator DIVIDE can be applied to the current container and one of S-collections belonging to the perspective. The DIVIDE operator is the complement of the Include (or JOIN) operator which was applied to the selected S-collection in order to include it into the perspective. Thus, for instance, if the user DIVIDEs

the current container shown in Figure 3b by the S-collection "Hyper-G", the scope of link-based browsing is reduced and navigation continues in the container shown in Figure 3a.

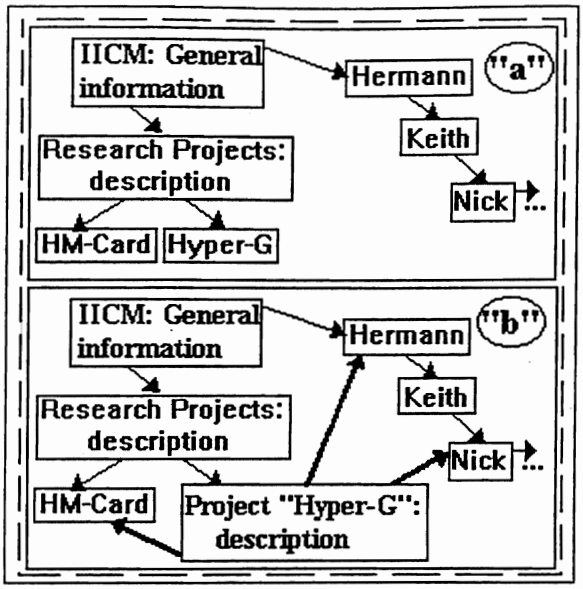

Figure 3Current container extended by the Include Operator

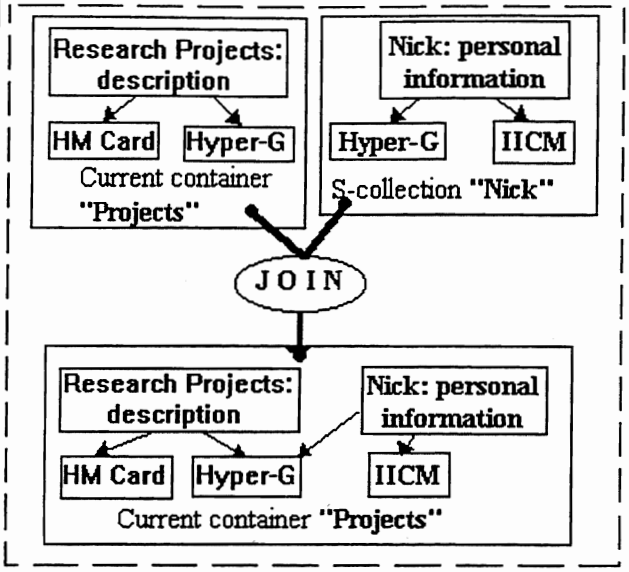

Figure 4: Current Container Extended by the JOIN Operator

In our prototype implementation, all the algebraic operators are available in the form of special buttons. Pressing a particular button invokes a list of S-collections which can be parameters of the operator in accordance with its semantics and current browsing situation. The user manually selects an S-collection which results in the modifications to the current container. Overview diagrams coupled with a display of the current perspective provide all the necessary feedback about the user's current location.

4. Concluding Remarks

The HM Data Model is a hypermedia data model based on abstract data objects called S-collections, which represent reusable "chunks" of hypermedia information (of any size). An S-collection may embody internal structure: hypermedia links are encapsulated within an S-collection and are bound between its member S-collections. The HM Data Model defines not only data structures (i.e. S-collections) per se, but also defines operations applicable to instances of such data structures and extends the default browsing semantics by introducing a new dimension of navigation.

The HM Data Model is distinct from other hypermedia data models in the level of granularity of the hypermedia data. Thus, S-collections are fully-fledged linkable and *navigable* objects. An S-collection encapsulates a particular link-based browsing strategy which is not mixed with strategies encapsulated within other S-collections (until users explicitly combine them by means of the algebraic operators). S-collections are well-defined units of interaction. A particular S-collection can be created, modified, deleted and/or reused as an independent abstract object via its public interface, without taking into account the internal structure of other S-collections or the hypermedia database as such. While a similar level of granularity is also provided by the Aquanet model, the authoring and browsing mechanisms described in this paper are unique to the HM Data Model.

The HM Data Model has been implemented as a prototype system for MS-Windows called HM-Card. The software, its documentation, and the implementation of the example described in this paper are available via anonymous ftp from "iicm.tu-graz.ac.at" in directory "pub\hmcard15".