Novi Sad J. Math.
Vol. 26, No. 1, 1996, 135–153

AN OBJECT-ORIENTED Hypertext SHELL

Mirjana Ivanović
Institute of Mathematics, Faculty of Science, University of Novi Sad
Trg Dositeja Obradovića 4, 21000 Novi Sad, Yugoslavia
e-mail:mira@unsim.ns.ac.yu

Abstract

A Hypertext shell is presented. It is a common shell which can be used in different fields and applications. The shell has been implemented using a specialized object-oriented programming language. Some advantages of this approach have been discussed.

AMS Mathematics Subject Classification (1991): 68N15
Key words and phrases: object orientation, Hypertext

1. Introduction

Fast development of methods and techniques in different fields of computer science determines the appearance of a great number of integrated multimedia systems for representation and presentation of information and knowledge. During the last several years, different concepts and directions of development systems for the representation and presentation of information can be distinguished. Most of these systems, however, possess the following basic elements (according to [12] and [8]):

• *Available knowledge (information, data)* - a set of objects and elements of real world domain, which have to be described and represented in a system. Information is easier represented if it is categorized and divided in small semantic entireties.

• *Common sense, supplementary knowledge* - additional explanations or supplement basic descriptions of the system domain.

• *Representation formalism* - basic and supplementary information and knowledge should be represented in an appropriate way. The system has to possess the formalism for suitable representation of information.

• *Presentation mechanisms* - represented information and knowledge can be used in different ways during the process of presentation. Different mechanisms have to support presentation in an adequate way.

• *Feed-back* - during the process of presentation, the system, in some instances, expects users' answers or reactions. The feed-back accepts them, converts them into appropriate actions and sends them to the system for further processing.[1]

The data which should be represented in a system for representation and presentation of data are usually complex. Thus different ways of multimedia representation like: text, picture, animation, sound, and so on are required. In modern systems, the data are too complex to be modelled with traditional database management systems (DBMS) and programming languages. The languages that come closest to providing the tools needed for a *comprehensive systems* fall under the category of object-oriented languages.

Programming of a system of this kind should usually have the following characteristics [6]:

• The object-oriented approach to data modeling and programming system.

• Persistent data objects should appear to the programmer no differently then transient objects do.

• The user should be able to create persistent data objects, copy them and use them in other situations.

• The programmer must be able to build and specify behavior for highly connected networks of objects, i.e. possibly arbitrary graphs of objects.

• There is no good reason to invent new programming language to do these things if there is one that comes near to providing the necessary se-

[1]In the rest of the paper we will use term **data** to denote information, knowledge, as well as data.

mantics.

Hypertext systems make a wide group of multimedia systems for representation and presentation of data from real world domain. On the basis of the available literature ([6], [11], [9], [3], [7]) and implemented Hypertext products, it can be noticed that:

• Hypertext systems are usually devoted to a special, concrete field,

• in the case that they use a programming language it is, usually, difficult (special language for picture management, special logical forms, languages based on the first order predicate calculus, ...) and only highly trained programmers can use them,

• the users can neither extend, nor modify existing system, nor build their own system(s).

The main idea explained in this paper is the possibility to the end-user to build, use, and extend Hypertext easily, clearly, and efficiently. For that reason an object-oriented Hypertext shell is proposed. It is based on an original, specialized object oriented programming language, which can be used by any non-computer professional. According to the main purpose of the language, it could not be an extension of some existing language, and it has to be designed as a brand new, highly specialized language *Less*.

In the following sections we describe the basic concept of Hypertext shell, basic structures and elements of *Less* language which are used to create Hypertext shell, and finally, an application of the proposed Hypertext shell.

2. Basic concept of Hypertext

Generally, Hypertext is a software tool for acquisition, storing, retrieval and presentation of data with references. Specifically, Hypertext can be defined as a data structure for representation of data, which includes a set of different tools for data maintenance and presentation. At its most basic level, Hypertext is a DBMS that enables connecting screens full of different data using associative links. At its most sophisticated level, Hypertext is a software environment for collaborative work, communication and knowledge acquisition [4]. The basic elements of every Hypertext, according to [5] are:

• *Database.* Database is a set of nodes with recorded data, methods of data recording and data access.

• *Scheme of node links.* Links connect the nodes into a semantic entirety and represent the way of moving through a Hypertext network of nodes. Every node is a source of at least one link with other Hypertext nodes.

• *Interface.* Interface supports visual presentation of data from nodes. It also enables simple movements from a node to some of the nodes directly connected with it.

Typically, three types of Hypertext that exist are:

- problem-resolving systems,
- on-line browsing systems,
- multi-purpose systems.

The Hypertext shell suggested in this paper enables implementation of different Hypertext applications. It is a general shell which can be applied in different fields and for different purposes. Filling in the Hypertext shell with specific contents and data from real world domains, a concrete Hypertext application is enabled. The Hypertext database consists of nodes. Every node is used for storing different kind of data - topics, and the nodes are interconnected according to different criteria. Links are the mode of transportation through Hypertext network.

The most appropriate structure for the representation of Hypertext database is a multi-digraph [8]. The graph nodes are Hypertext nodes - topics and the graph edges (links) are the paths through the Hypertext. There are three types of nodes: **starting**, **ending** and **inner** ones. Every Hypertext application contains one starting and at least one ending node. Different paths through the inner nodes enable different ways of moving through Hypertext and looking through data. The process of presentation and using Hypertext application consists of moving from topic to topic in some way and presenting data represented in the Hypertext database.

3. The environment for creating Hypertext shell

A primary goal of the Hypertext shell was to demonstrate that an integrated environment, built using object-oriented methodology, is the best environment to support hypermedia (hypertext) functionality. Similarly as in [7], our intention was to design and implement a general tool, mostly at a framework level, leaving only some application-specific details unresolved.

For the sake of simplicity, during the process of building a concrete Hypertext application, a specific object-oriented language *Less* is suggested. Wide range of end-users (non-computer professionals - like teachers, managers, and so on) can use the Hypertext shell and build their own Hypertext application. The main advantage of this Hypertext shell, in comparison with other similar systems, is that only minimal knowledge of computer technology is needed.

Less language has been suggested [1] as a basic environment in which the Hypertext shell is to be designed and implemented. The kernel of *Less* is a specialized object-oriented programming language which uses a lot of properties of object-oriented methodology. It consists of basic data structures, classes for representation of data, and statements and directives for using represented data. *Less* possesses mechanisms for creating new structures, filling them with data of various kinds, and using them in the process of presentation. *Less* has been designed as a tool for creation of different kinds of applications in the field of representation and presentation of data. The usage of *Less* to create Hypertext offers a possibility for creating a multi-purpose Hypertext shell which can be easily adjusted to specific needs. Furthermore, using the Hypertext shell and filling it with appropriate contents, concrete Hypertext applications are also easily achieved.

In the rest of this section, the elements of *Less* needed for building the Hypertext shell are described.

3.1 Built-in types and structures

Less has the following built-in data types (which are important for creation of Hypertext shell) [8]:

- `STRING` - a character array of limited length,
- `NUMBER` - arbitrary numerical value,

- `TEXT` - a formatted character array of arbitrary length,

- `PICTURE` - a graphical image (illustration),

- `ANIMATION` - an animated graphical presentation,

- `SOUND` - a sound,

- `IDENT ClassType` - determines a unique name of the object the type of which is `ClassType`,

- `PROCEDURE` - a private or public procedure, i.e. method included in a class. A private procedure is unique for every object client of the appropriate class. A public procedure is common for all object clients of the appropriate class.

Apart from these (built-in) data types, *Less* enables defining new types and structures by combining the existing ones.

`LIST OF Type` - a new structure - a list of an arbitrary number of elements which are of the same type `Type`.

$(name_1, name_2, \cdots, name_n)$ - a new unstructured type - enumeration of values which belong to this type. Values of that type are denoted by the n constant identifiers $name_1, name_2, \cdots, name_n$.

The language possesses a set of built-in classes. The most important classes for the Hypertext shell are `Text`, `Picture`, `Animation`, `Sound`, and `Info` which support storing different kinds of data: text, picture, animation, sound, and all of them in the same class, respectively.

3.2 Structure of *Less* program

Every *Less* program consists of three parts.

Part for the user-defined structures, where new classes and structures are defined.

Part for the user-defined procedures. Less implementation comes with a standard library attached. It consists of a lot of procedures which enable and support the process of creation and using of a Hypertext. The end-users can use them to build their own Hypertext without knowing any programming language or writing their own procedures. But, the user can (if possesses appropriate knowledge) redefine standard public procedures which participate in the process of presentation, and can define new, very specific, private procedures which can be treated as auxiliary procedures.

The first two parts are used to define different systems for representation and presentation of data.

Part for filling in multi-digraph, where every topic is filled in with appropriate contents. This part is equivalent to the phase of representation of data in Hypertext. In this part the user can use different kinds of directives and statements. Before the user approaches this phase, the data from real world domain is supposed to be split into topics, formalized and the links between them are supposed to be established. It is the last stage in the Hypertext creation. After that, the concrete Hypertext application is ready for use.

4. Structure of Hypertext shell

As mentioned earlier, a multi-digraph is the most appropriate structure for Hypertext. During the presentation of data, movements from one node (topic) to another are realized through different paths.

There are three basic kinds of topics in Hypertext. **Starting** topics are those in which the process of presentation of data from Hypertext has to start. **Ending** topics are those in which the process of using Hypertext can end. Coming to an ending topic indicates that a meaningful path through the multi-digraph was formed, and a meaningful presentation of data from Hypertext is over. **Inner** topics are visited during the process of presentation. They make a path from a starting topic to the ending one. Every inner topic is potentially an ending one, and the user of Hypertext can stop presentation in it.

Usually, there are a lot of different paths (ways) to get some determined ending topic from some determined starting one. Concrete path of presentation depends on the user of Hypertext, on his decisions and choices of the netx topic. In every topic the user should choose one of several possible topics, in which presentation will be continued.

Four possible ways of storing and representing of data - text, picture, animation, and sound, enable representation of different real world domains. Each of these means of storing and representing data, has its own specialties and can be described by a unique class. These classes can be defined and created using standard object-oriented mechanisms:

• *Class definition.* New classes are built by defining new instance variables and methods. Instance variables are of primitive data types or structures. These classes do not include behavior of any existing class.

• *Inheritance.* New classes inherit behavior (instance variables and methods) from other classes by inheritance mechanism. Instance variables and methods from inherited classes are directly absorbed in the new class.

• *Embedding (Inclusion).* New classes, also, can include other classes in their own definition. In that case, a new name is attached to the included class. All instance variables and methods of the included class are available only through that new name.

Everyone of the basic classes has the following simple structure:

```
TypeClass = CLASS { typeVar  : ARRAY OF type;
                    typeProc : PROCEDURE }
```

where

• `TypeClass` is one of `Text`, `Picture`, `Animation`, `Sound`, i.e. basic classes for representation of basic types of data in Hypertext.

• `typeVar` is one of `texts`, `pictures`, `anims`, `sounds` i.e. an instance variable which represents a list of elements for storing of different pieces of text, picture, animation or/and sound.

• `type` is one of `TEXT`, `PICTURE`, `ANIMATION`, `SOUND`, i.e. a basic type of data which can be stored in a Hypertext node.

• `typeProc` is one of `interText`, `interPict`, `interAnim`, `interSound`, i.e. a procedure which supports the process of presentation of special data type from topics.

Global class for storing all types of data is the class `Info`, in which classes `Text`, `Picture`, `Animation` and `Sound` are included.

```
Info = CLASS { ident :        STRING;
               key :          LIST OF STRING;
               tex :          CLASS Text;
               pic :          CLASS Picture;
               ani :          CLASS Animation;
               sou :          CLASS Sound;
               interAll :     PROCEDURE }
```

• `ident` - is a unique identification for every object of the type `Info`,

• `key` - is a list of key-words associated with the stored data.

• `tex, pic, ani, sou` - are included classes for storing different kinds of data.

• `interAll` - is a global procedure which specifies and supports the way of presenting data stored in Hypertext.

Multi-digraph, which is a framework for Hypertext shell, can be defined as a separate class. Class `HTS` specifies and determines the structure of multi-digraph in the Hypertext shell. A particular Hypertext application is an object of `HTS` class.

```
HTS = CLASS { ident     : STRING;
              topHTS    : LIST OF HTTopic;
              startHTS  : LIST OF IDENT HTTopic;
              stopHTS   : LIST OF IDENT HTTopic;
              trackHTS  : LIST OF IDENT HTTopic;
              currHTS   : IDENT HTTopic;
              interHTS  : PROCEDURE }
```

• `ident` - is a unique identifier of an `HTS` object.

• `topHTS` - is a list of topics constituting a database of Hypertext, i.e. a network of nodes. Every topic is an object of `HTTopic` class.

• `startHTS (stopHTS)` - is a list of unique identifiers of topics which are starting (ending) ones.

• `trackHTS` - is a list of topics identifications which were visited during the process of using Hypertext. They are chronologically ordered according to the appearance in the path. It is used when a skip to the previous node is needed.

• currHTS - during the usage of Hypertext we are always in one of the topics. currHTS contains the unique identifier of the current topic.

• interHTS - is a procedure which initializes presentation, supports the whole process of presentation, and ends it. This procedure is activated at the beginning of using Hypertext, it offers user a set of starting topics and expects him to choose a topic as a starting one. Then, it calls an appropriate procedure for data presentation from topics and guides the presentation until the end.

Every topic is an object of a class which has been defined as HTTopic class. HTTopic includes the basic class Info for representation of different kind of data.

```
HTTopic = CLASS {   ident  : STRING;
                    CLASS Info EXCL ident;
                    defaHT : NUMBER;
                    path   : LIST OF IDENT HTTopic;
                    ansHT  : PUBLIC PROCEDURE;
                    typeHT : (START,OBLEND,INNER) }
```

• ident - is a unique identifier for every object of type HTTopic.

• HTTopic inherits the basic class Info (without ident) for representation of data.

• defaHT - is the ordinal number of a key-word associated to the node where presentation will be continued by default.

• path - is a list of unique identifiers. To every key-word in the list (key) there is a topic in which presentation will be continued (if that key-word is chosen in current stage of presentation).

• ansHT - is a global procedure which should accept the user's "answer", i.e. identify the object which corresponds to chosen key-word (or return the default key-word).

• typeHT - determines type of a node (starting, ending or inner). Every node is also potentially ending if the corresponding list of key-words contains predefined key-word **END**.

The Hypertext's nodes, which are represented by `HTTopic` class, are typed nodes. They possess a unique identifier which uniquely determines every node. Connections (links) between the node and other nodes are specified by a `path` list. For every key-word in the topic, there is an appropriate topic name (unique identifier) in the list `path`. When a key-word is chosen during the usage of Hypertext, presentation is continued in the topic which is joined with the chosen key-word. A set of topics with joined `path` lists determines the Hypertext multi-digraph, e.i. the set of nodes and their links. The proposed classes make a multi-purpose Hypertext shell, which can be used for implementation of the different Hypertext applications. It defines simple mechanisms for creating, filling, in and using Hypertext in different areas.

5. An application of the Hypertext shell

As an example of using proposed Hypertext shell, we will give a small geographical Hypertext with fourteen topics (see fig. 1).

Node 14 is the obligatory ending node. When one reaches that node, it is not possible to move to another Hypertext node any more. There are several topics which can be chosen as starting nodes. If the user of Hypertext wants to use Hypertext without affecting the process of presentation (without choosing the key-word and the next topic), there is an appropriate default path for each starting node. This Hypertext application is a general form for the whole range of Hypertext applications which contain essential data of a country.

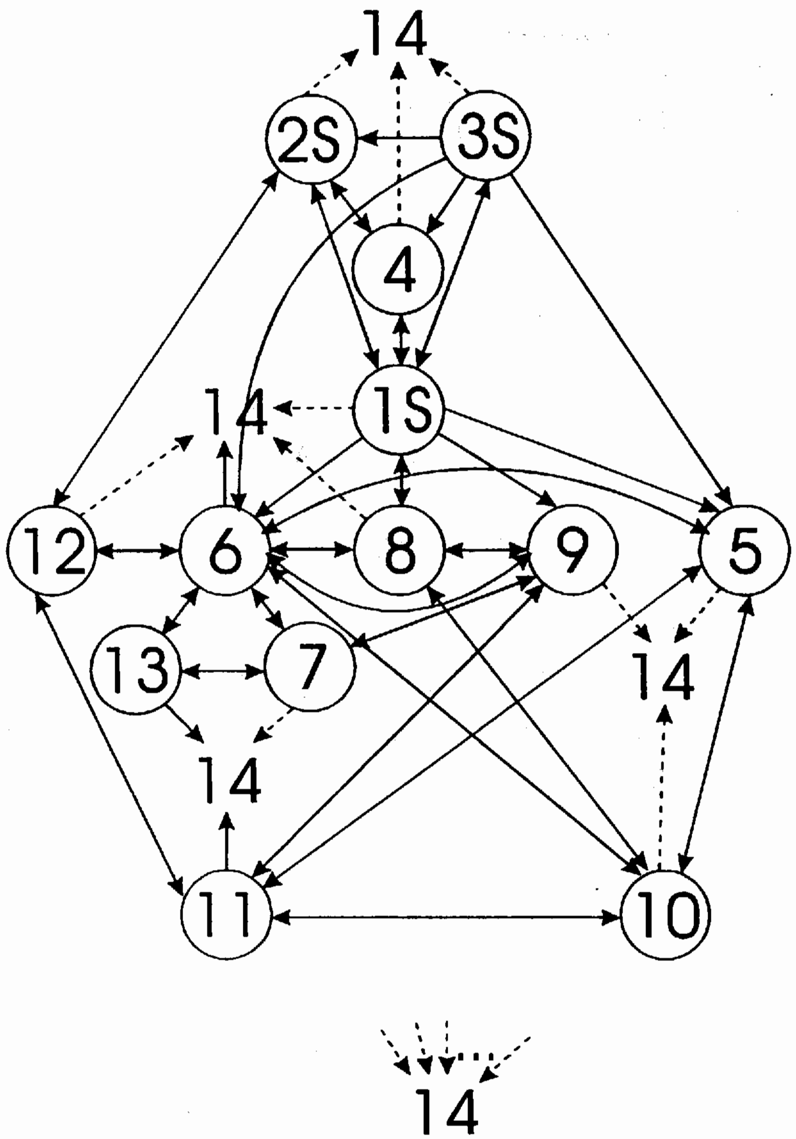

fig. 1

Concise descriptions of multi-digraph nodes are given.

1 (**starting**) - (**ident.**) basic data

- (**key-words**) continent history, country history, economy, demographics, manufacture goods, agricultural products, neighbors

- (**text**) basic data about country

- (**picture**) continent map with the country emphasized

- (**animation**) flag lifting

- (**sound**) national anthem

- (**default**) demographics

2 (**starting**) - (**ident.**) country history

- (**key-words**) continent history, basic data, culture

- (**text**) country history with key events

- (**picture**) set of country maps through history

- (**default**) culture

3 (**starting**) - (**ident.**) neighbors

- (**key-words**) continent history, basic data, country history, demographics, economy

- (**text**) basic data about neighbors

- (**picture**) part of continent map with the country and neighbors emphasized

- (**default**) economy

4 (**inner**) - (**ident.**) continent history

- (**key-words**) country history, basic data

- (**text**) basic data about continent history

- (**picture**) continent map through history

- (**default**) neighbors

5 (**inner**) - (**ident.**) demographics

- (**key-words**) education, economy, health statistics

- (**text**) population 1900-now, population growth, density doubling time, urbanization

- (**picture**) several charts which are connected with text

- (**default**) economy

All other nodes have similar contents and only their identifications are given in this paper.

6 (**inner**) - (**ident.**) economy

7 (**inner**) - (**ident.**) mining and quarrying

8 (**inner**) - (**ident.**) agricultural products

9 (**inner**) - (**ident.**) manufactured goods

10 (**inner**) - (**ident.**) health statistics

11 (**inner**) - (**ident.**) education

12 (**inner**) - (**ident.**) culture

13 (**inner**) - (**ident.**) energy

14 (**ending**) - (**ident.**) perspectives

The proposed Hypertext can be simply filled in with appropriate content, using *Less* language.

It can be noticed that from some nodes (6,11,13) a link leads to the obligatory ending node. But, as every node is potentially an ending node, there is a link ($->$) from every node to the ending one. The default paths from starting nodes are:

- for basic data (1s) - demographics (5), economy (6), agricultural products (8), health statistics (10),

- for country history (2s) - culture (12), education (11), manufactured goods (9), mining and quarrying (7),

- for neighbors (3s) - economy (6), manufactured goods (9), mining and quarrying (7), energy (13), perspectives (14).

Using the proposed structure of *Less* program (in 3.2.), a global program for described Hypertext is given. First two parts of the program are the Hypertext shell, while the third part is concerned with a particular Hypertext application. It enables filling Hypertext nodes with appropriate content and establishing links between the nodes.

```
PART CLASSES (* definition of necessary classes *)
HTTopic = CLASS { ... } HTS = CLASS { ... }

PART PROCEDURES
(* definition or redefinition of necessary procedures *)
PROCEDURE inter_HTS(HyTx : IDENT HTS);
(* supports the process of presentation *)
BEGIN
   First(&HyTx.startHTS, memst);
   WHILE NOT EndOfList(&HyTx.startHTS) DO
      WriteStr(memst); WriteLn; Next(&HyTx.startHTS,memst);
   END;
   ReadAns(&HyTx.currHTS); Choice = EMPTY;
   WHILE &HyTx.currHTS.typeHT <> OBLEND AND Choice <> END DO
      &HyTx.currHTS.interAll; &HyTx.currHTS.ansHT;
      &HyTx.currHTS = Choice;
   END;
END inter_HTS;
PROCEDURE InterALL(HyTx : IDENT HTS);
(* proc. for synchronizing display of all data from the topic *)
BEGIN (* display of textual data from the current topic *)
     First(&HyTx.currHTS.tex,T);
     WHILE NOT EndOfList(T) DO (* format it *)
          Display_TXT(T);
          (* hold it on the screen for some time *)
          Next(&HyTx.currHTS.tex,T);
          (* take the next text for display *)
     END;
     (* similar pieces of the code for other types of data *)
END InterAll;
PROCEDURE AnsHT;
(* procedure for acceptance of answer or key-word *)
BEGIN
```

```
     WHILE True DO
          ReadAns(Choice);
          IF string is read THEN
               nk = number_of_key_word;
               Choice = paHT[nk];
               IF key-word has been chosen THEN
                    RETURN
               END;
          ELSE (* take the default key-word *)
               Choice = paHT[defaHT];
          END
     END;
END AnsHT;

(* fill in Hypertext shell e.i. forming Hypertext application *)
START
(* Hypertext application with unique identifier COUNTRY *)
AssignOb(HTS, 'COUNTRY');
(* filling in Hypertext with the appropriate content *)
{  topHTS :  AssignList(TopicsCO);
           (* TopicsCO is the name of the list of topics *)
   startHTS : AssignMul(IDENT HTTopic, startCO,('basic
                 data','country  history','neighbors'));
   stopHTS : AssignMul(IDENT HTTopic, stopCO, ('perspectives'));
   currHTS : EMPTY;
   interHTS : Interpret(interHTS) };
(* making the first object in Hypertext *)
AssignOb(HTTopic,'basic data');
(* filling it with appropriate content *)
{  key : AssignMul(STRING,,('continent history', 'country
           history', 'economy', 'demographics', 'manufacture
           goods', 'agricultural products', 'neighbors', 'END'));
   tex : AssignTPAS(Text ... EndText);
   pic :  ani :  sou :
   interte : Interpret(DisplayTXT);
   interAll : Interpret(interALL);
   defaHT : AssNo(demographics);
   path : AssignMul(IDENT HTTopic,,('continent history',
           'country history', 'economy', 'demographics',
           'manufacture goods', 'agricultural products',
           'neighbors', 'END'));
   ansHT :   Interpret(ansHT);
   typeHT :  START };
(* adding of the first object in the multi-digraph *)
AddLast($TopicsCO,'basic data');
(* making of the second object in Hypertext *)
AssignOb(HTTopic,'country history');
(* filling it in with appropriate content *)
(* adding of the second object in the multi-digraph *)
AddLast($TopicsCO,'country history');
...
(* making and filling in others objects of multi-digraph *)
...
END.
```

There is another way of filling in Hypertext topics. In that way, all the necessary data must be prepared and stored in files. It does not influence the essence of design and implementation of Hypertext shell using *Less*, and will not be considered here.

6. Conclusion

The main goal of our research was to develop an object-oriented language suitable for creating and using different systems for representation and presentation of data (Hypertext, Authoring systems, ...). *Less* language serves as a "standard" for such applications and is a good tool for creating different kinds of systems.

By defining new classes suitable for a special kind of systems (in our example Hypertext) and developing various groups of procedures, we get a powerful shell for different specialized systems which belong to that group.

Classes and procedures for Hypertext shell constitute the first two parts of a *Less* program. By changing only the third part of the program (filling in the structure), different Hypertext applications are easily achieved.

The usage of *Less* in creating Hypertext offers a possibility for creating multipurpose Hypertexts which can be easily adjusted to specific needs. Furthermore, concrete Hypertext applications are also easily achieved, although not interactively.

The shell is easy to use, and a wide range of end-users can create and use their own Hypertext application with minimal knowledge of computers and programming languages and techniques.

References

[1] Budimac, Z. and Ivanović, M., On a specialized language for information (re)presentation, in Proceedings of I International Symposium DECSYM '92 (Side-Antalia, Turkey), 1992, 175-187.

[2] Carando, P., SHADOW - fusing Hypertext with AI, IEEE Expert, Winter 1989, pp. 65-78.

[3] Feiner, S.K. and McKeown, K.R., Automating the generation of coordinated multimedia explanations, IEEE Computer, October 1991, pp. 33-41.

[4] Fiderio, J., A Grand vision, BYTE, October 1988, pp. 237-244.

[5] Frisse, M., From text to Hypertext, BYTE, October 1988, pp. 247-253.

[6] Goodman, A.M., Harolick, R.M. and Shapiro L.G., Knowledge-based computer vision, IEEE Computer, December 1989, Vol. 22, pp. 43-54.

[7] Haan, B.J., Kahn, P., Riley, V.A., Coombs, J.H. and Meyrowitz, N.K., IRIS hypermedia services, CACM, Vol. 35, No. 1, January 1992, pp. 36-51.

[8] Ivanović, M., A Contribution to the Development of Programming Languages using Object-Oriented Methodology, Ph.D. thesis, University of Novi Sad, 1992.

[9] Meghini, C., Rabitti, F. and Thanos, C., Conceptual modeling of multimedia documents, IEEE Computer, October 1991, pp. 23-30.

[10] Nielsen, J., The art of navigation through Hypertext, Communication of ACM, March 1990, pp. 297-310.

[11] Pizano, A., Klingler, A. and Cardenos, A., Specification of spatial integrity constrains in pictorial databases, IEEE Computer, December 1989, Vol. 22, pp. 59-71.

[12] Wahlster W., Knowledge-based information presentation, CAS '90, Dubrovnik, September 1990.

Received by the editors June 22, 1995.