An Interaction Engine for Rich Hypertexts

Kasper Østerbye and Kurt Nørmark
Aalborg University
Denmark*

This paper was presented at the ACM European conference on hypermedia technology, ECHT'94, and obtained by ftp from ftp.iesd.auc.dk/pub/reports/papers/osterbye94c.ps.Z

Abstract

In semantically rich hypertexts it is attractive to enable presentation of a network of nodes and links at different levels of abstraction. It is also important that the user can interact with the hypertext using a command repertoire that reflects the chosen abstraction level. Based on a characterization of rich hypertext we introduce the concept of an interaction engine that governs the separation between internal hypertext representation and external screen presentation. This separation is the key principle of the HyperPro system. The HyperPro interaction engine is based on simple rules for presentation, interpretation of events, and menu set up. Much of the power of the interaction engine framework comes from the organization of these rules relative to the type hierarchy of nodes and links, and relative to a hierarchy of so-called interaction schemes. The primary application domain discussed in the paper is program development and program documentation.

Keywords: tailorability, interaction engine, aggregated views, event control, program development.

1 Introduction

The topic of this paper is development of interaction techniques on hypertexts that represent information from regular and structured domains. Our primary interest is hypertexts which represent source programs, program documentation, and similar information captured during program development or maintenance. However, the interaction techniques we have developed are of interest to developers of a much broader set of hypertexts, which we will call "rich hypertexts".

In *rich hypertexts* all the nodes have types. The type of a node may reflect its role in the hypertext, e.g. its syntactic category. The nodes in rich hypertexts are connected by typed links. In many application domains it may in addition be important to ensure that a rich hypertext obeys a set of topological constraints. A structure may be imposed on the contents of nodes as well. Thus, the "richness" stems from the typing of both nodes and links and from the degree of structuring at the microscopic level[1] (inside nodes and links) as well as at the macroscopic level (among nodes and links).

Rich hypertexts may appear in a variety of different application domains. One such domain is engineering as discussed by Malcolm *et al.* in [14], and illustrated by the systems Dynamic Design [1], Ishys [8], and the present paper. Another domain is structured argumentation, as represented by gIBIS [4], Authors Argumentation Assistant [22], and AquaNet [15]. In [16] Nanard and Nanard use the term *rich hypertext structure model* to cover essentially the same notion.

In most hypertext systems it is satisfactory to show only one node in one window. In our work, however, we emphasize the creation of coherent, textual views on several nodes and links. Such views typically consist of aggregation of text from several interlinked nodes. We strive for independence between (1) the internal node and link representation and (2) the presentation of the hypertext on the screen. Hereby the views can be adapted or customized, without altering the underlying node and link representation. In order to enhance the readability we also find it important to control the textual decoration (fonts, colors, and indentation) of each constituent in a view.

In order to gain practical experience with our ideas we have built an experimental system called HyperPro. The HyperPro system provides an extensible data model and powerful means for tailoring the interaction with user-defined nodes and links.

We organize the node and link types in a hierarchy with inheritance. The top of the hierarchy is *EntityType*, of which *Node* and *Link* are subtypes. *EntityType* specifies that attribute-value pairs can be attached to all entities. Links are one-

*Department of Mathematics and Computer Science, Fredrik Bajers Vej 7E, 9220 Aalborg Ø, Denmark. (Internet: kasper@iesd.auc.dk and normark@iesd.auc.dk). This research was supported in part by the Danish Natural Science Research Council, No. 11-0061.

[1] The microscopic level corresponds to the within component level in the Dexter model of hypertext [11]. The macroscopic level corresponds to the storage level in that model.

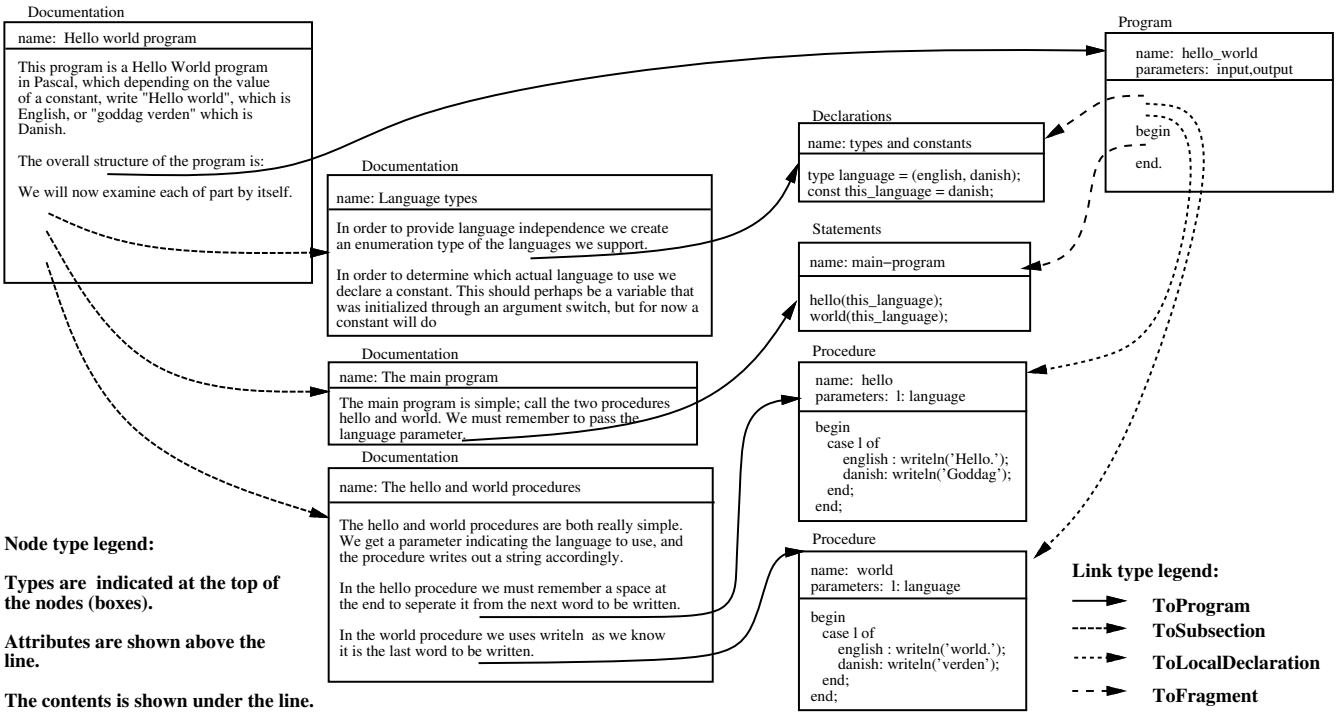

Figure 1: *A rich hypertext with documentation and program fragments.*

to-one, directed, and can be anchored in the source node.[2] Nodes are entities which have contents and a name. All types may specify required attributes, which are attributes that the system will guarantee to be associated with all instances of the type. Required attributes are inherited from supertypes. The data model is further discussed in [24].

In this paper we will propose a framework called an *interaction engine*, which makes it possible to define and control the interaction on rich hypertexts. The interaction engine is controlled by a set of *interaction rules* which are associated with the entity types and with a new kind of abstraction which we call an *interaction scheme*. Throughout the main section of the paper (section 2) we will illustrate our work with hypertexts which represent programs in a software development process. We begin with an example that explains how literate programming [13] can be supported by interacting with a rich hypertext. Following this, we describe and discuss the interaction engine concept. Finally, we summarize the most relevant similar work in section 3.

2 The interaction engine

An *interaction engine* is the component of a hypertext system, which mediates the interaction between a user and the hypertext, via directives specified in a dedicated interaction specification language. The HyperPro interaction engine is controlled by a set of interaction rules formulated in a simple, special-purpose interaction specification language.

Before describing the design of the HyperPro interaction engine in details, we will highlight some of its functionality through examples. Following this we will proceed with a description of the underlying concepts and mechanisms of the interaction engine.

2.1 Using the interaction engine

In this section we will illustrate the capabilities of the HyperPro interaction engine, by giving a number of examples where it is used in the area of program development. The examples will be drawn from a hypertext that interrelates a Pascal program and accompanying documentation in the spirit of literate programming.

The main purpose of the interaction engine is to enable a multitude of interactions (presentation and event interpretation) on a single underlying hypertext. In figure 1 we show a sample rich hypertext with four documentation nodes and five program nodes. The program nodes are of four different types: *Program*, *Procedure*, *Statements*, and *Declarations*. There are a number of different relationships between the nine nodes in the figure, which are represented using different link-types. The example is a simple “hello world”

[2] We have not yet implemented anchoring of links in the destination nodes.

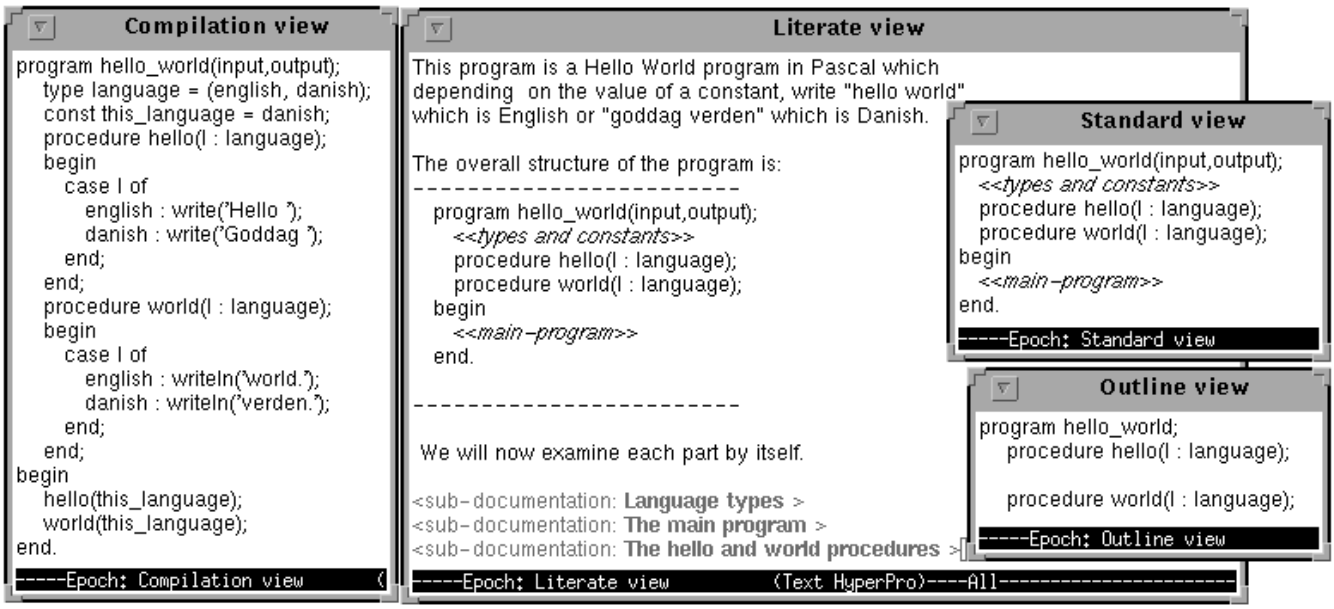

Figure 2: *Four different presentations of the same program.*

program[3], which has the twist that it can write the message in both English and Danish.

Depending on the situation we may be interested in presenting the program fragments of the hypertext from figure 1 at different levels of abstraction and from different perspectives. At the highest abstraction level we might be interested in a procedural outline of the program, showing only which procedures are defined at what scope level. An example of such an *outline view* is shown in figure 2 (lower right). At the most concrete level, we certainly need a full-detail presentation which can be given to the Pascal compiler. This *compilation view* of the program is shown in figure 2 (left side). In between these it is possible to define some other useful presentations. Figure 2 (top right) shows a presentation which we call a *standard view* of the main program. A standard view shows one node in full detail and the immediate outgoing link context in less detail. All the presentations in figure 2 are generated by application of HyperPro interaction rules. In essence, the applied rules decide in which detail the anchored links of a node should be presented.

Using the compilation interaction scheme, the destination node of the anchors is shown in-lined in full detail (which implies that had there been local procedures of 'hello' or 'world', these would be shown as well). We are able to control the indentation when presenting an in-lined node. If a procedure is shown in a window by itself, it does not have to be indented. This is because we chose to control the amount of indentation as an aspect of the presentation of the link leading to the procedure.

In the standard interaction scheme (which gives the standard view mentioned above), we specify that only the signature of the procedure nodes should be presented. Structurally the presentation is similar to that produced by the compilation scheme. In the compilation view, we create presentations of the destination nodes of the four outgoing links from the program, and we put them at the anchor points of the outgoing links. We do the same in the standard view, but here the presentations of the destination nodes contain less information. It should be noted that we have organized the textual contents of a node in a number of attributes, which allows for easy access to the important constituents, such as procedure names and parameter lists (see figure 1). The direct access to such attributes is useful to create tiny presentations of nodes, as in the standard views. Also note that presentation rules contribute with some text constants, such as the keyword "procedure" and other lexical items. In [17] we give a full account of the issues related to internal structuring using attributes and anchoring.

Following the idea of literate programming [13] we can also present the program together with its documentation as in figure 2 (center). Here we begin with the root documentation node (the node in the upper left corner in figure 1), and specify that links of type 'ToProgram' should be presented as a standard view of the destination node. We have also specified that program text should be shown slightly indented relative to the surrounding documentation. Links to subsections of the documentation are shown as "`<sub-documentation:` **subsection-title** >" (At the bottom of the literate view). This is done by presenting each of the three links of type 'ToSub-

[3] The only reason to chose the "hello world" example is that it is simple enough to fit in the limited space of a paper like this, and good enough to demonstrate our interaction approach with rich hypertexts.

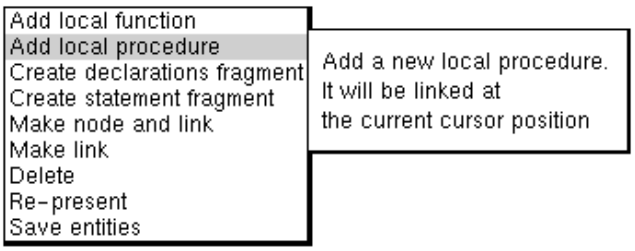

Figure 3: *The popup menu for the standard view of programs.*

section' from figure 1 in a label interaction scheme, which in turn presents the destination nodes using the name attributes only. We have specified that these short texts are shown in boldface and that they are purple (the color cannot be seen in the figure).

Besides being able to present the underlying hypertext in different ways, an interaction engine should also be able to control and facilitate commands from the user. In a bare hypertext system, it is typically sufficient to support a relatively small set of generic commands, such as a command that creates a new node and links it to the current focus point in some other node. In systems for rich hypertexts it is desirable to support a broader set of commands which depend on the node and link types on which they are issued.

In the Pascal program and documentation example given above, it would, for instance, be fairly natural to have a command that creates a new local procedure and links it to the program. Notice that this command can have built-in knowledge of the types of the node and link involved, and that this frees the user from being prompted for this information when the network is extended. Our interaction engine provides facilities for defining menus and for binding menu entries and keystroke sequences to underlying operations that manipulate the structure of the hypertext. The menus available depend on the context, so that only meaningful menu entries will appear.

Figure 3 shows the menu that has been associated with the standard view of programs. The menu will appear in the literate view of figure 2 when the menu-button is activated over the program text, because that part of the text constitutes a standard view of the program. If the user hesitates in selecting an entry, the help text next to the menu appears.

As a final important aspect, the presentations of nodes and links can be edited. However, we might want to limit the editability, for instance such that programs are only editable in the literate view (in order to support the discipline that documentation and and program are written together). The editor keeps track of which parts of the text belong where in the underlying hypertext, and it is able to store the text back into the internal representation.

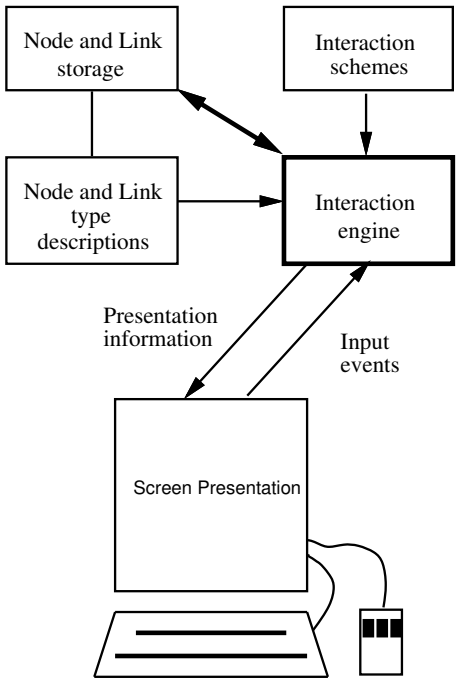

Figure 4: *The architecture of an interaction engine based system.*

2.2 Description of the interaction engine.

The HyperPro system has been designed for tailorability at two levels: (1) At the data model level, where we can specialize the predefined node and link types, and (2) at the user interface level, where we can specify the interaction (presentation and event interpretation) on rich hypertext. This paper explores only the second issue.

We will now examine the inner workings of the interaction engine. The overall architecture of the system is shown in figure 4. As can be seen from the figure, the engine is controlled by the actual nodes and links, the types of these, and the interaction schemes.

The interaction engine is able to *locate* and *effectuate* interaction rules. Locating interaction rules amounts to a lookup-process based on information about the entity on which we operate, and on the interaction scheme to be used. Effectuation amounts to the execution of some action on the underlying hypertext, typically a generation of a presentation, or a structural change of the hypertext.

A rule consists of an *enabling condition* and a *consequence*. The enabling condition states under which circumstances the rule can take effect, and the consequence specifies what action to carry out, or which value to return if and when the rule is selected for execution. We say that a rule is *applicable* on some entity E if the enabling condition holds when evaluated on E.

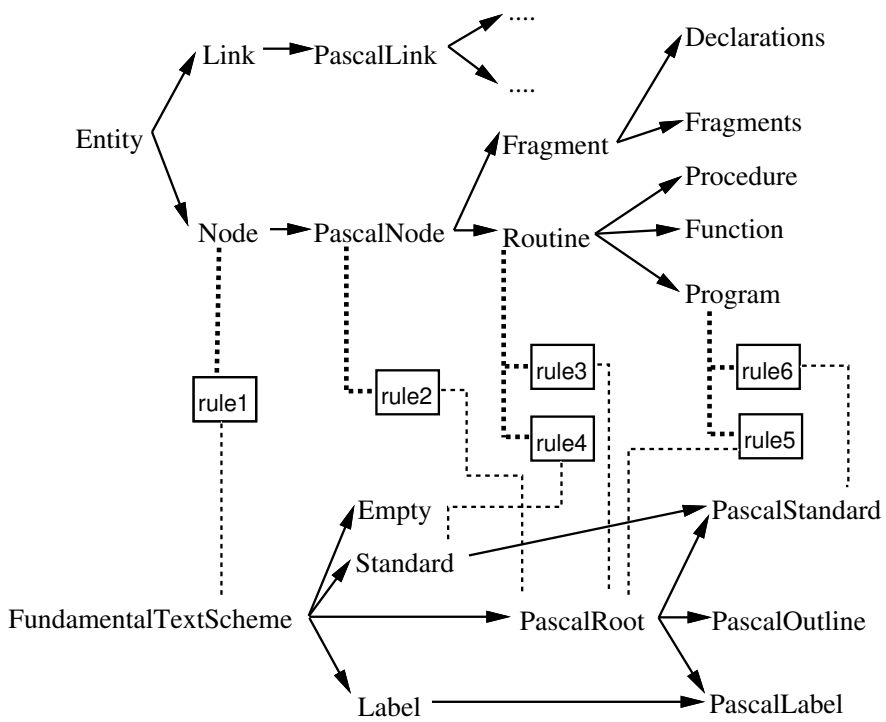

Figure 5: *Entity type and interaction scheme hierarchies together with interaction rules.*

Each interaction rule is associated with an entity type. (This is similar to the association of operations to types in object-oriented programming languages). An interaction rule which is associated with an entity type ET is implicitly also associated with the subtypes of ET. This is inheritance in the hierarchy of entity types.

In a system based on rich hypertext we can expect many interaction rules explicitly or implicitly associated with each entity type. Therefore we categorize the interaction rules in two orthogonal dimensions: (1) According to the *kind* of the rule (whether the rule governs presentation, event interpretation, or menu set up), and (2) according to the *role* of the rule in the dialogue with the user (whether, for instance, the rule is concerned with outline views, standard views or compilation views, as discussed in section 2.1). In HyperPro, the role is captured by the notion of interaction schemes. We now define an interaction scheme as a named set of interaction rules which may involve rules associated with different entity types and rules of different kinds.

It can be observed that the interaction rules in different schemes are unlikely to be disjoint. Typically there will be some interaction rules which we have to put into several interaction schemes. In order to avoid a duplication of rules we also organize the interaction schemes in an inheritance hierarchy (here with the possibility of multiple inheritance).

Putting these things together, we end up with an overall picture as illustrated in figure 5. We show a number of interaction rules associated with the hierarchies which are relevant for the example from section 2.1. We see that each interaction rule is associated with both an entity type and an interaction scheme from each of the two hierarchies. Since more than one rule may be attached to each pair of type and scheme the rules on such a pair are ordered. The HyperPro interaction engine implements the inheritance in the entity type and interaction scheme hierarchies by a dynamic rule lookup-process. The result of the lookup-process is a totally ordered list of applicable interaction rules, the first of which is considered the most specific, and the last the most general.

If we want to present (or otherwise manipulate) an entity of type *Program* in the interaction scheme *PascalStandard* we first look at all the rules on the entity *Program* (rule5 and rule6). Rule6 is more specific than rule5 because rule6 is associated with a more specialized scheme than rule5. Next we look at the rules on *Routine* (rule4 and rule3). For the same reasons as above, rule4 is more specific than rule3. Provided that all the rules are applicable, and that they are of a relevant kind, we end up with the following ordering of the rules:

rule6, rule5, rule4, rule3, rule2, rule1.

These rules are ordered lexicographically, based on topo-

logical sorts of *PascalStandard* and its super schemes[4], and *Program* and its supertypes. The inspiration for this ordering strategy comes from the object-oriented programming language CLOS [2]. As can be seen, we give first priority to entity types, and second priority to the interaction schemes. Thus, when finding the mutual order of two rules $r1$ and $r2$, we only take the association to interaction schemes into account if the rules are associated with the same entity type.

In the following we will discuss presentation rules, event rules and menu rules, and we describe how the interaction engine effectuates rules of these three kinds.

2.2.1 Presentation rules

The presentation rules of nodes and links give us the following basic presentation possibilities:

1. Textual aggregation text constants and attributes of an entity.
2. Recursive activation of the presentation framework at neighbor entities.
3. Control of the graphical appearance (font, size, indentation, foreground color, and background color) of each piece of text.
4. Control of the read/write protection of each attribute.

If we take a look at the literate view in figure 2, we will examine the rules that are necessary to produce it. The interaction scheme used is called *LiterateInteraction*. The presentation is specified as a rule, which is attached to a *Documentation* node type and the *LiterateInteraction* scheme. The rule looks as follows:

```
NodePresentationRule
  @True ->
  <{for ToProgram use LiterateInteraction
    for ToSubsection use DocumentationLabel
    for CrossReference use DocumentationLabel
    for others use inherited}>
```

The enabling condition of the rule is "True". The rule states implicitly that the main textual contents of the node should be presented, and that the listed interaction schemes should be used to present links, which are anchored in that text. As can be seen, links of type *ToProgram* should be presented using the *LiterateInteraction* scheme, and links of type *ToSubsection* and *CrossReference* should be presented using a scheme named *DocumentationLabel*. *Label schemes* are typically used in connection with other interaction schemes, to prevent the entire hypertext from being shown. By convention schemes which are named Label should only produce short presentations in the order of a few words.

When doing the rule-lookup, only the most specific, applicable rule is applied to generate the textual presentation. However, in the above rule we use a clause "for others use inherited". The `others` part refers to links which are not of type *ToProgram*, *ToSubsection*, or *CrossReference*, and the `inherited` specifies that we apply less specific rules (relative to the ordering of rules described above) for the presentation of anchored links of other types.

The rule associated with the link type *ToProgram* and the *LiterateInteraction* scheme is the following:

```
LinkPresentationRule
 @True ->
 <'------------------\n'>
 <destination node in PascalStandard
     indent: 3>
 <'------------------\n'>
```

This presentation rule describes that three pieces of text must be aggregated. First a dashed line is drawn (with a newline at the end). Then the destination node of the link is shown in the interaction scheme *PascalStandard*, and this text should be indented 3 units. Finally one more dashed line is drawn. Notice the advantage of having the indentation and the dashed lines in a link rule, as opposed to in a (destination) node rule: The presentation of the program node depends on the link type, from which it is reached.

In the above examples we have shown how we can specify (1) the presentation of the main contents of a node and its anchored links, (2) the usage of text constants (such as the dashed lines), and (3) text generated from the destination node of a link. Besides these, we can include values of attributes of both nodes and links, and we can show incoming and outgoing links from a node independent of their anchoring. This is particularly useful when producing outlines where we do not want to present the actual contents of any nodes, but merely appropriate labels of each node.

2.2.2 Event rules and menu rules

As mentioned in the Pascal example it is attractive to be able to tailor the semantics of the commands, as issued via menus and via the keyboard. The HyperPro interaction engine allows us to specify event handling and to define menu entries.

When an event is generated on a presentation, it is our approach to locate an applicable event rule, which defines an action on the underlying network of entities. In order to make this work, we must have an entity type and an interaction scheme for the rule-lookup. Both the entity type and

[4] The direct super schemes are ordered in the definition of an interaction scheme. In the example *PascalStandard* inherits from both *Standard* and *PascalRoot*, and because of the ordering, the scheme *Standard* comes before *PascalRoot*. Hereby, the topological sorting of the scheme *PascalStandard* and all its direct and indirect super schemes is *PascalStandard*, *Standard*, *PascalRoot*, and *FundamentalTextScheme*.

the interaction scheme, which was used for creation of the presentation, can be determined from the state of the editor. This, in turn, makes it possible to ensure that only meaningful entries appear on the menus, and that events are interpreted adequately.

Each menu rule specifies a single menu entry. In order to assemble a full menu on some specific part on a presentation, we do a rule-lookup, and assemble a menu of the entries from *all* applicable menu rules. As shown in figure 3 of the Pascal example we have created menu entries for addition of new local procedures etc. By selecting a menu entry, a high-level event is generated. The event is resolved by an additional rule-lookup, but now looking only for (the most specific of the) event-rules. The consequence of an event is a call to a predefined interaction primitive of the interaction engine.

The interaction engine has a number of predefined primitives. The most central of these are:

- *MakeNodeAndLink* creates a new node and links it to the current focus (a node and an anchor point), as maintained by the editor. The operation takes either none or three arguments. The three arguments are the type of the new node, the type of the new link, and the name of the interaction scheme to be used for presenting the new link. If no arguments are given, the user is prompted for this information.
- *MakeLink* creates a new link from the current focus to an existing node. This operation takes none or two arguments analogous to the case above.
- *Re-present* presents the current focus in another window in another interaction scheme. The routine takes none or one argument, the interaction scheme for the new presentation. Again, if the argument is not given, the user is prompted for a scheme name.
- *Redirect* the target of a link.
- *Delete* an entity.

It is worth noticing that we have no routine for following a link. In HyperPro "following of link" amounts to a representation of the link, or the destination node of the link, in another interaction scheme (which typically shows more details). In many hypertext systems "following" is a command issued at link markers. In our system there is no link markers as such, but instead brief presentations of the entities, which usually are generated using label schemes. When we define a new label scheme it is good practice to provide a menu item and an event rule that defines how to "follow".

3 Similar work

The work on HDM [9] is similar to our work in several ways. The authors of HDM are focusing their attention on hypertext systems for *authoring-in-the-large*, which characterizes hypertext systems that are used to produce what we call rich hypertexts. In [9] the necessity of a hypertext modeling system is also strongly argued. A main focus of HDM has been to develop modeling capabilities for hypertext at the storage level. In HyperPro we have attempted to extend the modeling capabilities to the interaction aspect of the system through interaction rules and schemes. The closest thing to our interaction schemes in HDM is *perspectives*. Perspectives are used to control which parts of an entity are to be shown. Perspectives are subsumed by our notion of interaction schemes, as are the HDM notion of binding anchor types to link types. To show an entity from different perspectives corresponds somewhat to showing it using different interaction schemes. We are able to specify how attached links are to be shown in a given interaction scheme, achieving the same as binding anchor types to link types. In HDM one can also indicate that links of a given type are not to be shown, by *not* binding any anchor type to the unwanted link type. In HyperPro this is done by specifying that the unwanted link type should be presented in an interaction scheme which produces the empty text string.

HyperPro can be discussed in terms of the Dexter architectural vocabulary from [11]. In the Dexter model, the instantiation of nodes and links depends on a presentation specification, which is considered as primitive in the model. We see our notion of interaction schemes as a concrete proposal for how to elaborate and further develop the Dexter presentation specifications for rich hypertext, so as to obtain tailorability in the user interface. In addition, the interaction engine approach addresses how to interpret events on a hypertext presentation. This is not an issue in the Dexter model.

In the "seven issues" paper [12] Frank Halasz *et al.* discusses a problem with the document compiler from Notecards. A document compiler extracts information from the hypertext, and combines it into a linear text intended for printing. The problem is that the linearized document becomes a *new* entity in the system, and corrections made to the linearized document are not reflected back into the hypertext. Another problem in Notecards is the lack of mechanisms allowing the user to view a document at multiple levels of abstraction. Our interaction schemes seem to solve both of these problems. We can define interaction schemes that will extract information and combine it just as a document compiler will do, and the combined text can be shown in the text editor or exported to an external tool. However, the text can still be edited, and changes will be saved in the appropriate nodes. We are also able to produce several levels of outlines through different interaction schemes.

MacWeb [16] has powerful mechanism for synthesizing documents. The document synthesis is specified in a dedicated language WebTalk (used for other purposes as well), which seems to be more powerful than our simple rule language. However, it seems like our system has two properties that the

MacWeb system is lacking: First, changes to the synthesized node is not reflected back into the original nodes, but rather the synthesized node becomes a node in its own right. A synthesized node corresponds to a *presentation* in HyperPro, but changes to presentations are reflected in the internal *representation*. Second, it is not clear to what extent MacWeb allows tailoring of the command repertoire.

The Grif structured document editor described in [19] is the Hypertext system which seems to resemble HyperPro the most. In Grif the logical structure of a document is described by a document type definition (DTD) which describes attributes and the hierarchical structure of the basic entities. Separated from the logical structure, it is possible to create presentation models that are similar to the HyperPro interaction schemes, in that they describe how different attributes and subcomponents are to be presented on the screen, including such aspects as font and color. The system also allows for several simultaneous views of the same logical structure (for instance showing it at an outline level and a detailed level at the same time). Editing done in one view is reflected in other views as well, as it is the same underlying structure being manipulated. In HyperPro we do not update the views immediately, but would like to do so in the future. Grif solves the two problems mentioned by Halasz in the same way as HyperPro does.

There are, however, also notable differences between Grif and HyperPro. First, Grif distinguishes between hierarchical structure and links, and links come in two main varieties: cross-reference links and inclusion links. Inclusion links are treated as hierarchical structures, the destination node can be shown in-lined at the anchor point, and a Grif presentation can make full use of the inheritance scheme for the Grif presentation model. The HyperPro interaction schemes treat all links as inclusion links, but in HyperPro some inclusions are very short. The second and most prominent difference is, however, that Grif uses presentation models which can only control the appearance, while HyperPro interaction schemes have control over both appearance and input events.

The main inspiration for our notion of interaction schemes comes from the area of structure-oriented programming environments [5]. In many of these environments (such as GANDALF [10]) so-called presentation schemes (also known as unparsing schemes) are used to specify how syntax trees are pretty printed on the screen. In our work we have generalized these schemes to interaction schemes, and we have adopted them to a hypertext framework. As in HyperPro, the separation between the internal representation and the external views is also important in structure-oriented programming environments. Similarly, the question of structure-editing versus text-editing plays an important role in these environments. In HyperPro we have settled on text editing at the entity level, and structure-editing at the overall level. HyperPro may be seen as a basis for a structure oriented environment based on an internal graph representation. As such, it is relevant to compare it with IPSEN [6], which is a programming environment based on graph grammars. IPSEN is based on a fine grained graph representation, whereas in HyperPro we rely on more coarse grained nodes.

Finally, it is relevant to point out some other work on the border between hypertext and programming environments: HyperWeb [7] is a framework which supports the creation of hypertext-based software development environments; DOgMA [21, 18] is a hypertext tool for documentation and maintenance of C++ programs; similarly, HyperSoft [20] is about program comprehension enhanced by hypertext browsing tools.

4 Conclusion

In this paper we have characterized a kind of hypertexts, which we call rich hypertexts. In summary, rich hypertexts are well-typed, and with a high degree of structuring internally in the hypertext entities, and among the nodes and links. We have demonstrated that in rich hypertexts it is possible to define user interfaces, which are more elaborate than the user interfaces of most existing hypertext systems. In our approach, the user interface is defined and controlled by a number of simple rules, which are organized with respect to both the types of links and nodes, and with respect to interaction schemes.

In summary, the rule-base interaction engine framework in HyperPro makes it possible to aggregate and decorate textual pieces from neighbor entities in a rich hypertext, and to present the result in a window on a graphical screen. In addition, it is possible to control the user interaction on such presentations, by means of exactly the same rule-lookup technique as is applied for the generation of views.

It is possible to make detailed as well as more abstract presentations of rich hypertexts. The abstract presentations are clearly the most interesting of these, because they can be used to convey some kind of overview of a non-trivial subset of the hypertext in a single window on the screen. One kind of abstract presentation is *outlines*, as known from many existing text editors and word processors. We have identified another kind of abstract presentation, which we call *aggregated presentations*. In an aggregated presentation, information from two or more nodes (typically, but not necessarily of different types) is presented together, but kept apart in the underlying hypertext. (Aggregated presentations are also recommended in the work on MacWeb [16]). The literate view (see figure 2) is an example of an aggregated presentation. In [25] (an extended version of the present paper) we have shown how the interaction engine can be used to emulate much of the interaction of gIBIS [4].

HyperPro is an experimental system, a prototype, which we use to get practical experience with our ideas. As of now (summer 94), the most significant part of HyperPro is the interaction engine, as described in this paper. The storage

level in HyperPro is realized by the Fenris HyperBase [3] (a locally developed hyperbase, which is part of our ongoing line of hyperbase research, briefly described in [23]). The HyperPro kernel, including the interaction engine, is implemented in Smalltalk. We have textual HyperPro interfaces running in both Smalltalk and in Epoch (which is an Emacs-18 derivate). The Hyperbase and the Epoch-based hypertext editor are connected to the Smalltalk kernel via simple network protocols, which have been developed as part of the project.

References

[1] James Bigelow and Victor Riley. Manipulating sorce code in dynamic design. In *Proceedings of the HyperText'87 conference*, pages 397–408, 1987.

[2] Daniel G. Bobrow, Linda G. DeMichiel, Richard P. Gabriel, Sonay E. Keene, Gregor Kiczales, and David A. Moon. Common Lisp object system specification X3J13 document 88-002R. *Sigplan Notices*, 23(Special Issue), September 1988.

[3] M. Boel, A. Gregersen, P. Larsen, and F.H. Møller. Manual og dokumentation til fenris version 2.0. Technical Report IR 91-02, Department of Mathematics and Computer Science. Fredrik Bajers Vej 7E, 9220 Aalborg Ø, Denmark, 1991. (In Danish).

[4] Jeff Conklin and Michael L. Begeman. gIBIS, a hypertext tool for exploratory policy discussion. *ACM Transaction on Office Information Systems*, 6(4):303–331, October 1988.

[5] Susan A. Dart, Robert J. Ellison, Peter H. Feiler, and A. Nico Habermann. Software development environments. *Computer*, pages 18–28, November 1987.

[6] G. Engels, C. Lewerenz, M. Nagl, W. Schäfer, and A. Schürr. Building integrated software development environments part I: Tool specification. *ACM Transaction on Software Engineering and Methodology*, 1(2):135–167, April 1992.

[7] James C. Ferrans et al. Hyperweb: A framework for hypermedia-based environments. In Herbert Weber, editor, *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments*. Software Engineering Notes, 17(5):1–10, December 1992.

[8] P. K. Garg and W. Scacchi. Ishys - designing an intelligent software hypertext system. *IEEE Expert*, 4(3):52–63, Fall 1989.

[9] F. Garzotto, D. Schwabe, and P. Paolini. HDM - a model based approach to hypermedia application design. *ACM - Transactions on Information Systems*, 11(1):1–26, January 1993.

[10] A.N. Habermann and D. Notkin. Gandalf: Software development environments. *IEEE Transaction on Software Engineering*, pages 1117–1127, December 1986.

[11] Frank Halasz and Mayer Schwartz. The Dexter hypertext reference model. *Communication of the ACM*, 37(2):30–39, February 1994.

[12] Frank G. Halasz. Reflections on NoteCards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836–852, July 1988.

[13] Donald E. Knuth. Literate programming. *The Computer Journal*, May 1984.

[14] Kathryn C. Malcolm, Steven E. Poltrock, and Douglas Schuler. Industrial strength hypermedia: Requirements for a large engineering enterprise. In *Proceedings of the Hypertext'91 conference*, pages 13–24, 1991.

[15] Catherine C. Marshall, Frank G. Halasz, Russel A. Rogers, and William C. Janssen Jr. Aquanet: A hypertext tool to hold your knowledge in place. In *Proceedings of the Hypertext'91 conference*, pages 261–275, 1991.

[16] Jocelyne Nanard and Marc Nanard. Using structured types to incorporate knowledge in hypertext. In *Proceedings of the Hypertext'91 conference*, pages 329–343, 1991.

[17] Kurt Nørmark and Kasper Østerbye. Representing programs as hypertext. In Boris Magnusson, Görel Hedin, and Sten Minör, editors, *Proceedings of Nordic Workshop on Programming Environment Research*, pages 11–24. Lund Institute of Technology, Lund University, Sweden, 1994. LU-CS-TR:94-127.

[18] Gustav Pomberger. DOgMa: A tool for the documentation and maintenance of software systems. Technical report, Department of Software Engineering, Institut fur Wirtschartsinformatik und Organisationsforschung, Johannes-Kepler-Universität Linz, 1991.

[19] Vincent Quint and Irène Vatton. Combining hypertext and structured documents in Grif. In *Proceedings of the ACM ECHT Conference*, 1992.

[20] Airi Salminen, Jussi Koskinen, and Jukka Paakki. Hypersoft: An environment for hypertextual software maintenance. In Boris Magnusson, Görel Hedin, and Sten Minör, editors, *Proceedings of the Nordic Workshop on Programming Environment Research*, pages 25–37. Lund Institute of Technology, Lund University, Sweden, 1994. LU-CS-TR:94-127.

[21] Johannes Sametinger and Gustav Pomberger. A hypertext system for literal C++ programming. *Journal of Object Oriented Programming*, pages 24–29, January 1992.

[22] Wolfgang Schuler and John B. Smith. Author's argumentation assistant AAA: A hypertext-based authoring tool for argumentative texts. In A. Rizk, N. Streitz, and J. André, editors, *Hypertext: Concepts, Systems and Applications*, pages 137–151. INRIA, Cambridge University Press, November 1990.

[23] Uffe Kock Wiil. Research lab news: Hyperbase reserach at Aalborg University. *Siglink Newsletter*, 1(2):12–14, 1992.

[24] Kasper Østerbye. Structural and cognitive problems in providing version control for hypertext. In D. Lucarella, J. Nanard, N. Nanard, and P. Paolini, editors, *ECHT'92, Proceedings of the ACM Conference on hypertext*, pages 33–42, 1992.

[25] Kasper Østerbye and Kurt Nørmark. Tailorable interaction using the HyperPro interaction engine. Technical Report R94-2017, Aalborg University, Department of Mathematics and Computer Science. Fredrik Bajers Vej 7E, 9220 Aalborg Ø, Denmark, April 1994.