AALBORG UNIVERSITY

Tailorable Interaction Using the HyperPro Interaction Engine

by

Kasper Østerbye and Kurt Nørmark

R-94-2017 ISSN 0908-1216 April 1994

INSTITUTE FOR ELECTRONIC SYSTEMS
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE
Fredrik Bajers Vej 7 — DK 9220 Aalborg Ø — Denmark
Tel.: +45 98 15 85 22 — Fax.: +45 98 15 81 29

Tailorable Interaction Using the HyperPro Interaction Engine

Kasper Østerbye and Kurt Nørmark
Aalborg University*

Abstract

In this paper we identify and characterize a kind of hypertext, which we call rich hypertext. We are most interested in application of rich hypertext in the domain of program development, but other areas are considered as well. The separation between the internal rich hypertext representation and the external screen presentation of the hypertext is a key principle, on which this work is based. We introduce the interaction engine concept, which governs the separation between the hypertext representation and the screen presentation. An 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 the type hierarchy of nodes and links, and relative to a hierarchy of so-called interaction schemes.

1 Introduction

The topic of this paper is development of interaction techniques on hypertexts which represent information from regular and structured domains. Our primary interest is hypertexts which represent source code, program documentation, and similar information captured during a program development process. However, the interaction techniques, which 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 syntactic category, or its role in the hypertext. The nodes in rich hypertexts are connected by typed links. There is a structure imposed on the contents of nodes as well as links. In many application domains it may in addition be attractive to ensure that a rich hypertext obeys some topological constraints, which involve the types of the node and link instances. As a minimum, it is often useful to formulate constraints that limit the source and destination node types for links, and the types of attached links for nodes. 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 *structured argumentation*, as represented by gIBIS [5], Authors Argumentation Assistant [20], and AquaNet [16]. Another domain, in which rich hypertexts are present, is engineering as discussed

*(Internet: kasper@iesd.auc.d and knormark@iesd.auc.dk)

[1] The microscopic level corresponds to the within component level in the Dexter model of hypertext [10]. The macroscopic level corresponds to the storage level in that model.

in the paper "Industrial Strength Hypermedia: Requirements for a large engineering enterprise" [15], and illustrated by the systems Dynamic Design [3], and Ishys [8].

In most hypertext systems it is satisfactory to show only one node in one window together with the attached and anchored links. In our work, however, we emphasize the creation of coherent, textual views from several nodes and links. Such views typically consist of aggregations of text from several interlinked nodes and, in turn, from the attributes of each contributing link and node. Our approach can be seen as a contrast to the use of hierarchical browsers (as used in HyperWeb [7] and DOgMA [19]), through which each node or attribute is presented in relative isolation from each other. We are furthermore striving for a notion of independence between the internal node and link representation and the hypertext presentation on the screen. Hereby external views can be adapted or customized, without altering the underlying node and link representation.

As mentioned above, the nodes and links of rich hypertexts have types. We find it important that the set of node and link types can be easily extended and adapted to each new application area. We organize node and link types in one common type hierarchy: the entity type hierarchy. Our basic hypertext model is defined by a few general (top level) entity types which can be specialized so as to fit new areas of applications. This is in contrast to those hypertext systems, in which a fixed set of node and link types has to be used in all possible situations and applications. The developers of a rich hypertext system for a new application area should be able to delineate important structuring principles during the hypertext modeling phase. In addition we find it attractive to let the interaction, as experienced by end-users of rich hypertexts, depend on the type of the entities, which are involved. This is in contrast to the hypertext systems, in which all interaction follow some fixed patterns that are applied to all entities. In our work we propose to associate the rules, which govern the definition of external views as well as the interpretation of events, to the types of nodes and links.

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*. The interaction rules are attached to node and links types, as well as to a new kind of abstractions which we call *interaction schemes*. It is the task of the interaction engine to localize and execute the relevant interaction rules as a result of the users work on a rich hypertext. Some interaction rules define how nodes and links are presented in a window; others define the effect on events from the mouse, the keyboard, and menus.

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 start with an example that explains how literate programming [12] can be supported by interacting with a rich hypertext. Following this, we describe and discuss the interaction engine concept. The material in this section of the paper can be understood as the basis of structure-oriented programming environments [6], in which the primary internal program representation is the network formed by nodes and links. In section 3 we will give an example from an entirely different domain on the use of the interaction engine. The example, which is taken up, is how to simulate the interaction functionality of the gIBIS system [5] Finally, we summarize the most relevant similar work in section 4.

2 The interaction engine

An *interaction engine* is a 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 "driven" by a set of interaction rules, which are formulated in a simple, special-purpose interaction specification language. Some rules define a view on a set of nodes and links, hereby determining the external appearance of the hypertext at the user interface level. Other rules control the interpretation of events, which are issued by the user on a hypertext view.

The interaction rules are organized relative to the types of the nodes and links. In addition, the rules are associated with so-called interaction schemes, the purpose of which is to classify the interaction rules relative to various user interface objectives. Given some specific interaction context (view generation, menu setup, or event interpretation), the interaction engine will locate the relevant interaction rules, in the order from the most specific rule to the most general rule, which are applicable in the situation. Depending on the interaction context, a single rule (the most relevant and specific), or all the rules, are effectuated. Seen in isolation, each interaction rule is dedicated to a single well-defined interaction task. Much of the power from the HyperPro interaction engine framework comes from the organization of the rules, which we mentioned above, and which will be discussed in further details below.

The use of an interaction engine is seen as an alternative to the toolkit approach, such as in [17], in which the interaction with hypermedia entities are governed by a set classes (a framework), programmed in a general-purpose, object-oriented programming language. We see several advantages of the interaction engine approach over the toolkit approach. First, the use of a special purpose interaction specification language instead of a general purpose programming languages raises the abstraction level of the user interface description. We can expect a more clear and concise formulation of the user interface via the rules of an interaction engine than through the classes in the toolkit approach. Second, the user interface defined through an interaction engine is expected to be more tailorable than a user interface made via the toolkit approach. At any time it is possible to add or redefine a few interaction rules in the engine, or to create a new interaction scheme. This provides for incremental changes of the interaction, relative to existing rules and schemes. At no time, it is necessary to learn the inner workings of the engine, nor to master some general programming language, in order to tailor the interface of the hypertext system.

The advantage of the toolkit approach compared with the interaction engine approach is that the toolkit allows for a greater spectrum of user interfaces. At the moment HyperPro only supports textual presentations, but we expect the interaction engine approach to be useful for presentation of other media-types as well.

Before we go into any details on how we have designed the HyperPro interaction engine, we will highlight some of its functionality through an example. Following that 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 give a number of examples illustrating the capabilities of the HyperPro interaction engine, used in the area of program development. The examples will be drawn from a rich 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 and presentations on a single underlying hypertext. In figure 1 we have shown a sample rich hypertext with four nodes of documentation and five nodes of program. The program nodes are of four different types: program, procedure, statement-fragment, and declaration-fragment. In the nodes of types 'procedure' and 'program' we have, in addition shown the internal structuring in terms of attribute-value pairs. (The text under the line of division in these nodes is the main content of the nodes). There are a number of different relationships between the nine nodes, represented

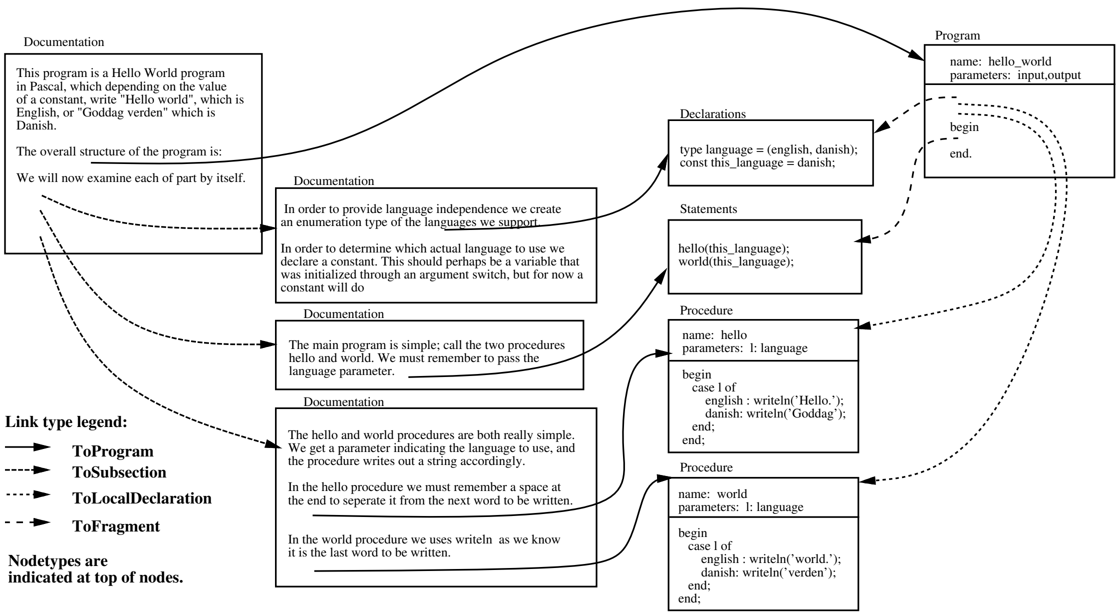

Figure 1: *A rich hypertext with documentation and program fragments.*

using different link-types. The example is a simple "hello world" program[2], which has the twist that it can write the message in both English and Danish.

Depending on the situation we can be interested in presenting the program fragments of the hypertext from figure 1 at different levels of abstractions. At the highest abstraction level we might be interested in a procedural outline of the program, only showing 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 example 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 an example, 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 tells in which detail to present the anchored links of a node.

Using the full scheme, the destination node of the anchors are shown in-lined in full detail (which will imply 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, which leads to the procedure.

In the standard scheme (which gives the standard view mentioned above), we specify that only the signature of the procedure nodes should be presented. The presentation is structurally very

[2]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.

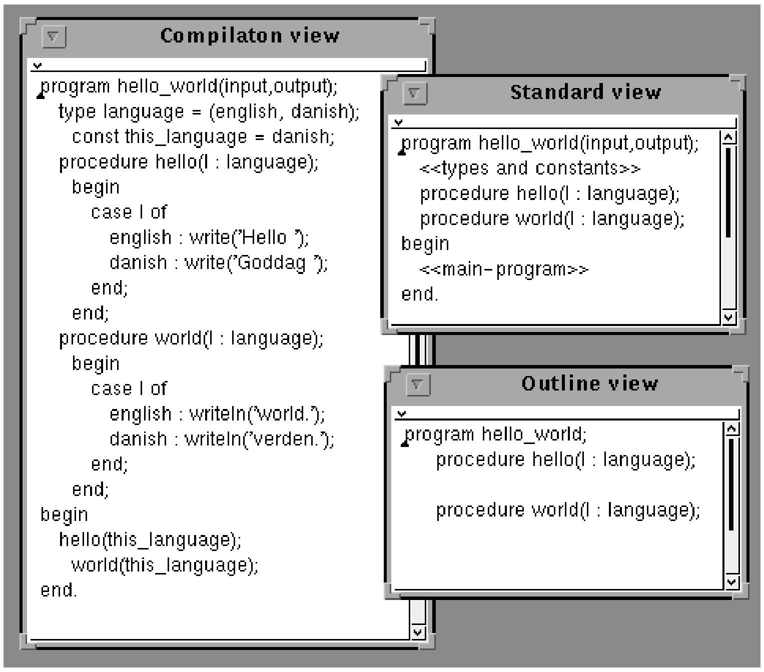

Figure 2: *Three different presentations of the same program.*

similar to that produced by the full scheme. In the full 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 noticed that we in our elaboration of rich hypertext organize the textual contents of a node in a number of attributes, which allow us easy access to the important constituents, such as procedure names and parameter lists - see figure 1. The direct access to such attributes is very useful to create tiny presentations of nodes, as in the standard views. Also notice that that presentation rules contribute with some text constants, such as the keyword "procedure" and other lexical items.

Following the idea of literate programming [12, 2] we can also present the program together with its documentation as in figure 3. Here we have taken outset in the root documentation node (the node in the upper left corner in figure 1), and specified 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* `>`". This is achieved by presenting each of the three links of type 'ToSubsection' from figure 1 in some label interaction scheme, which in turn presents the destination nodes as the title attributes only. As part of the elaboration we have specified that these short text are shown in boldface and that they are purple (which sadly cannot be seen on the figure).

Besides being able to present the underlying hypertext in a flexible number of 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 relative small set of generic commands, such as a command that creates a new node and link it to the current focus point in some other node. In

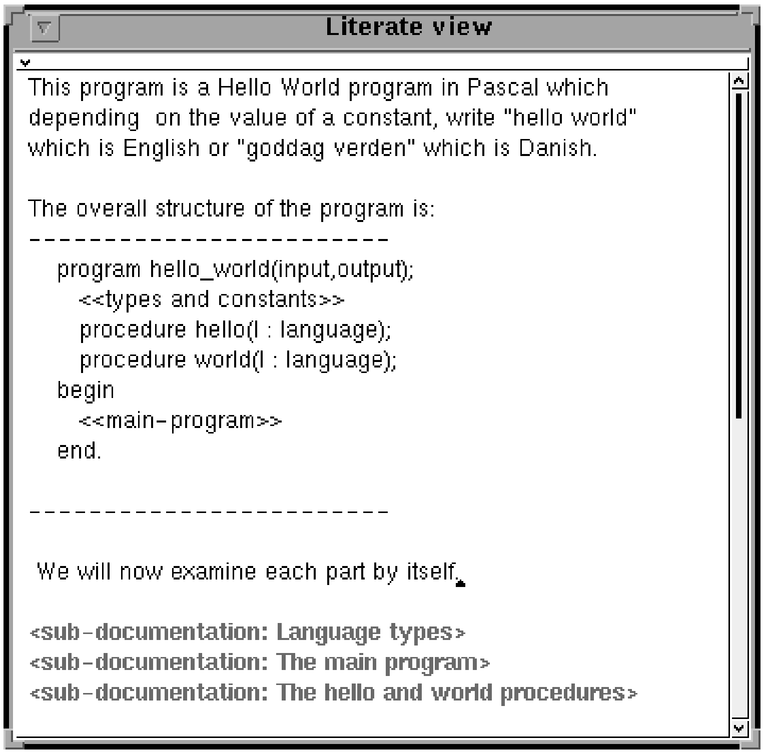

Figure 3: *A literate view where documentation is shown together with the program.*

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 entered.

In the Pascal program and documentation example from 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 also provides facilities for defining menus and to bind menu items 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 items will appear.

In figure 4 is shown the menu that have been associated with the standard view of programs. The menu will appear i the literate view of figure 3 when the menu-button is activated over the program text, as that part of the text constitutes a standard view of the program. When the user hesitates in selecting an item, 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 promote that code and and program are written together). The interaction engine keep track on which parts of the text belongs where in the underlying hypertext, and be able to store the text back into the internal representation.

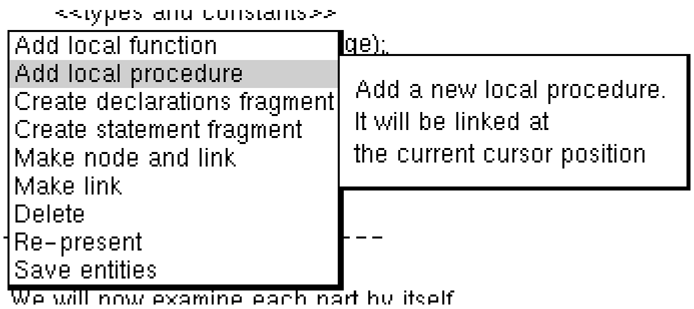

Figure 4: *The popup menu for the standard view of programs.*

2.2 Description of the interaction engine.

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

In this section we will examine the inner workings of the HyperPro interaction engine. The overall architecture of the system is shown in figure 5.

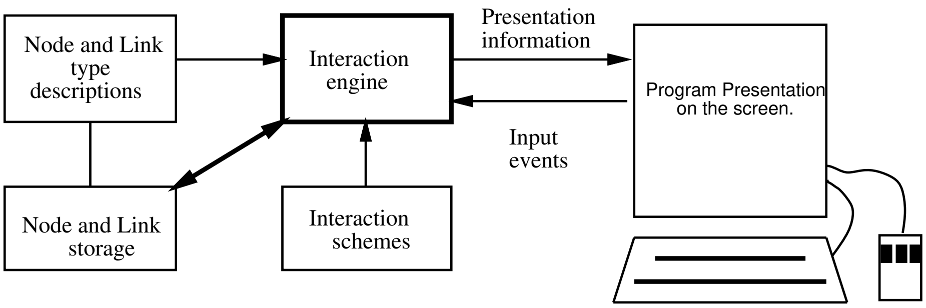

Figure 5: *Architecture of an interaction engine based system*

As can be seen from the figure, the engine is controlled by the nodes and links, the types of the nodes and links, and the interaction schemes. An *interaction scheme* is a plan of how to carry out interaction on the set of entities in a program network. An interaction scheme is defined by a set of interaction rules. These rules—and hereby the scheme—determine "input and output" characteristics of the involved entities. An *interaction rule* is associated with both an entity type and an interaction scheme. The rule prescribes how an entity of a given type functions in accordance with a given interaction scheme. Some rules define the presentation of entities, while others define input response.

The interaction schemes are organized in a specialization hierarchy, where each scheme (apart from a distinguished root scheme) has one or more super schemes. This hierarchy makes it possible for an interaction scheme to *inherit* rules from its super scheme(s). Below we will explain the semantics of the rule inheritance in terms of a rule lookup algorithm on entity types and interaction schemes.

We have defined a number of general interaction schemes in HyperPro, the rules of which gives HyperPro a conventional hypertext “look and feel”. Following the rules in these general schemes, one node is presented in one window, all outgoing links are presented as simple link markers, and there are commands for creating and deleting links and nodes at a relatively low level. When we make specialized rich hypertexts for some new application domain, we may both specialize the entity types and the interaction schemes. The specialization of interaction schemes also provides for customization of the overall interaction with the hypertext.

We will now take a closer look at the overall organization of entity types and interaction schemes, as well as the rule lookup process. Assume we want to present an entity of type D according to interaction scheme U. We first examine whether there is any presentation rule directly associated with the interaction scheme U and the entity type D. If there there is, that rule is used as a prescription of the presentation. If not, we will somehow take the rules on the super types of D and on the super schemes of U into account, and thereby gaining the inheritance effect on the rules, as mentioned above.

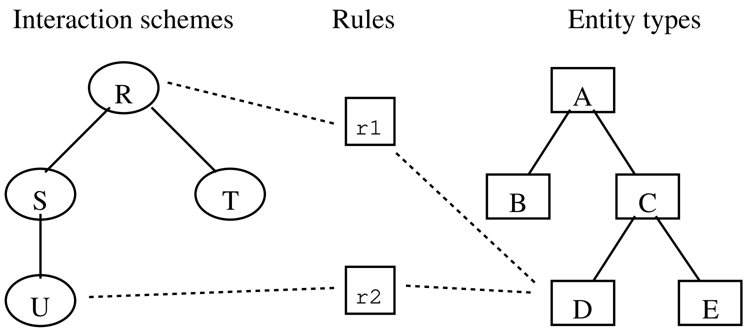

Figure 6: *Entity type and interaction scheme hierarchies.*

In order to illustrate the rule lookup process in more details we will assume that we have the entity and interaction scheme hierarchies as shown in figure 6 with two associated interaction rules. If we want to present an entity of type D in scheme U, we will search for rules in the following order: (U,D), (U,C), (U,A), (S,D), (S,C), (S,A), (R,D), (R,C), (R,A). This is the lexicographic ordering, made out of some specific topological sorts of U and its super schemes, and D and its super types. As can be seen, we give first priority to interaction schemes, and second priority to the entity types. Thus, when finding the mutual order of two rules $r1$ and $r2$ we only take rules of the entity types into account if the interaction scheme does not resolve the mutual ordering. In the example, $r2$ precedes $r1$.

Each 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 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. When we perform a rule lookup, we only take applicable rules into account. Rules that are attached to the exact same entity type and interaction scheme pair are rated mutually, with the aid of an additional ordering on the rules that belong to an entity type. In that way, the result of the lookup process is a totally ordered list of applicable interaction rules. In some interaction situations we only need the most specialized rule, in other situations we need them all.

In the following we will discuss presentation rules, event rules, and menu rules in further details.

2.2.1 Presentation rules

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

1. Textual aggregation of attributes from an entity.
2. Control of the graphical appearance (font, size, indentation, foreground color, and background color) of each attribute.
3. Control of the read/write protection of each attribute.
4. Control of the editing mode of each attribute.

If we take a look at the literate view in figure 3, 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 DocumentationLabelScheme
    for CrossReference use DocumentationLabelScheme
    for others use inherited}>
```

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 DocumentationLabelScheme. *Label schemes* are typically used as part of other interactions schemes to prevent the entire hypertext to be 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 a 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 at 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 PascalStandardScheme
     indent: 3>
 <'------------------\n'>
```

This rule describes that tree 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

PascalStandardScheme, and this text should be indented 3 units. Finally one more dashed line is drawn.

In the above examples we have shown how we can specify the presentation of the main contents of a node and its anchored links, the usage of text constants (in the form of dashed lines), and text from the destination node of a link. Besides these, we can include values of attributes of both nodes and links, and we can show in 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 items.

When an event is generated on a HyperPro 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 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. To assemble a full menu on some specific part on a presentation, we do a rule lookup, and assembles a menu of the items from *all* applicable menu rules. As shown in figure 4 of the Pascal example we have created menu entries for addition of new local procedures etc. By selecting a menu item, a high-level event is generated. To resolve an event, we perform 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, 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 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 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.

As explained in the Pascal example, the need to prompt users for types of links and nodes are usually eliminated when we are in a situation where we know the types we are dealing with.

In the Pascal example the menu item "make local procedure" activates an event, which is then translated into a call to MakeNodeAndLink with the appropriate parameters.

It is worth noticing that we have no routine for following a link. The following of a link is really a special case of re-presentation. 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 Examples of use from other domains

The purpose of this section is to demonstrate that the interaction engine ideas are useful beyond the domain of program development. We will demonstrate that by showing how and to what extent we are able to simulate the gIBIS system described by Conklin and Begeman in [5]

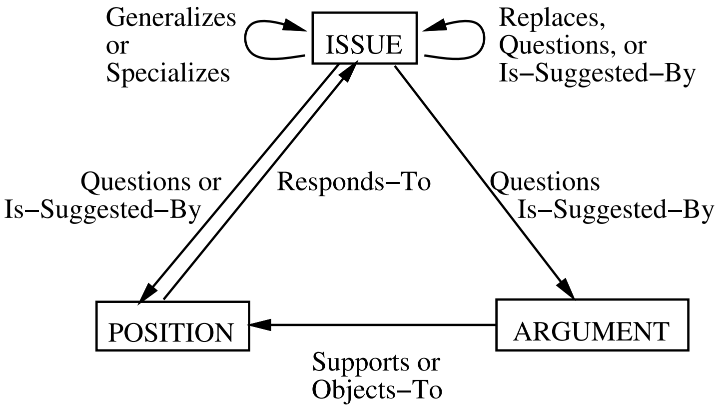

Figure 7: *The set of legal rhetorical moves in IBIS.*

The gIBIS system is designed to aid a number of designers structure their design discussions. As all discussions take place using the computer, the system will therefore capture their design rationale. The gIBIS system is based on a rhetorical model illustrated in figure 7 (adapted from [5, page 305]). The general idea is that in a design there are a number of *issues* to which people responds to with positions, which others then supports or objects to using arguments.

The gIBIS system contains an advanced textual browser. HyperPro does only support textual presentations, so we can not create a graphical browser. Consequently we will examine how we can adapt HyperPro to support the IBIS method in a text only fashion, where the interface provides as much support as possible.

As an exercise, we have created a number of entity types which support the gIBIS method. A gIBIS hypertext, created from these entity types, is an example of a rich hypertext. There are three different node types which can be related in eight different ways using different link types, and there are strict topological constraints (e.g. that an argument cannot support an issue). Each node type has an internal structure in the form of attributes, such as author, date, keywords etc. We have also created a number of interaction schemes, which allow us simulate the interaction of gIBIS. The presentation in figure 8, as well as the menu functionality on it, have been generated

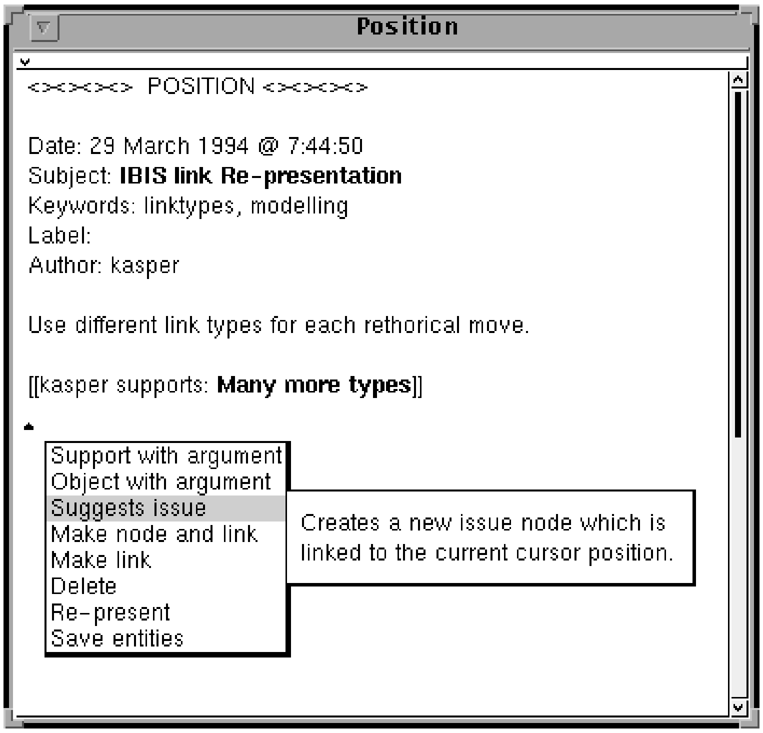

Figure 8: *Full presentation of a position and a menu.*

by means of the HyperPro interaction engine.

- The text seen in figure 8 was produced by the use of a presentation rule. The rule specification make use of text literals such as the "position" line at top, "Date:", "Subject:" and newlines. The rule presents attributes of the node, such as the actual date and subject. Finally the rule uses the contents and anchor part, to produce the text body with embedded anchors. The "link markers" differ depending on the types of the links. For the link shown, a support link, we have chosen to create the link marker from several text fragments too. First we enclose the link marker in "[[" and "]]". We then get the author name of the destination node (this is again a presentation scheme which only returns the author of the node), the text literal "supports:", and finally we get the name of the destination node (in bold face).

- As opposed to enclosing the link marker in brackets and having a literal that says "supports", we could have used colors to mark the same information. We can control both the text color and the background color. Thus we can to some extent simulate the usage of colors as done in gIBIS. The use of background colors are particularly attractive as it gives the link markers a button-like appearance.

- Unlike the gIBIS system, it is not necessary for us to parse the node in order to restore changes in the text body or to extract the subject or keywords. The editor will at all times keep track of which text fragments belong to which attributes in which nodes. Furthermore we can indicate as part of the rule which text fragments can be edited and which cannot.

As an example, one cannot edit the author field, whereas the subject field can be edited.

- As can be seen in the figure, we have set the system up with a menu to create and manipulate the IBIS structure. From the menu for positions there are commands to create a new supportive or objective argument, or to raise a new issue. The action of these menu commands draw upon lower level commands that will create nodes and links. When the user hesitates with a menu selection, a helptext is displayed next to the current menu selection. The helptext is given as part of the description of a menu item.

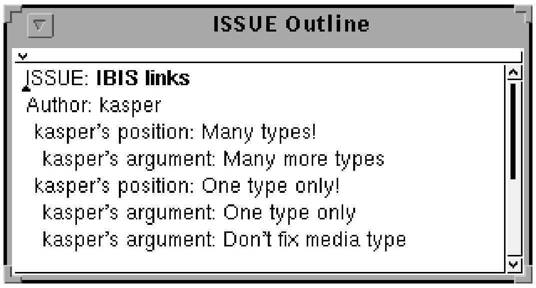

Figure 9: *Issue outline presentation.*

In the gIBIS system, a graphical presentation together with a node index view (a small view with a single line for each node) are used to provide a user with means to grasp the overall structure of an issue. Text can also be used rather efficiently to provide overviews. In figure 9 we have shown the same issue as before. The text in the figure is generated by means of an outline scheme, which just shows the name of the issue and its original author and the positions and arguments regarding the issue. The actual text of the issue is not shown, but links to positions are presented by applying the outline scheme recursively, and indenting each position.

The outline scheme presents us with an interesting problem. When we present information from links which are anchored, without showing the actual text they are anchored in, the editor will not be able to maintain anchor positions for us. In the outline in figure 9, this means that we can not create a new position, because we can not determine where in the issue-text it should be anchored. This is unfortunate, because such an operation would actually make a lot of sense.

In the concrete case we might solve the problem by not anchoring the links at all, but the general problem remains. Currently we "solve" it by not allowing new links to be created (by excluding the commands to do so from the menu). Notice however, that the titles of the nodes in the outline can be edited as usual, and that changes will be reflected back into the hypertext.

4 Similar work.

The work on HDM [9] is very similar to our work in many ways. The authors of HDM are focusing their attention on hypertext systems for *authoring-in-the-large*, which characterize 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 in HDM to our interaction schemes are *perspectives*. Perspectives are used to control which parts of an entity is 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 no link marker.

HyperPro can be discussed in terms of the Dexter architectural vocabulary from [10]. 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 of 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 [11] Frank Halasz discusses a problem with the 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 lack of mechanisms which allows the user to view a document at multiple levels of abstraction. Our interaction schemes seems 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. 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. KMS [1] suffers from the same problems as in Notecards. The KMS answer to the document editing problem is to allow the compiled document to be shown in a view-only mode, so all changes must take place in the normal frames.

The Grif structured document editor described in [18] 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 hierarchical structure of the basic entities. Separated from the logical structure, it is possible to create presentation models, which are very much like the HyperPro interaction schemes, in that they describe how different attributes and subcomponents are to be presented at the screen including such aspects as font, color etc. The system also allows for several simultaneous views of the same logical structure (for instance show 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, noticeable differences between Grif and HyperPro. First, Grif distinguishes between hierarchical structure and links, and links come in two main varieties, cross-referential and inclusion. Inclusion links are treated like hierarchical structures, the destination node can be shown inlined at the anchor point, and a Grif presentation can take full use of the inheritance scheme for Grif presentation model. Cross references are treated differently which seems to give them some problems. In HyperPro we do not have an a priori distinction between link types. The HyperPro interaction schemes treat all links as inclusion links, only some inclusions are very short - typically containing only the name of the destination node, but as shown in figure 2 also more complicated presentations, here the procedure heading. 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 from both keyboard and mouse. We can thus tailor the command repertoire to include specialized commands that reflect the document types we are working with.

It is our hypothesis that HyperPro can used as a basis for a structure-oriented programming environment [6]. One of the fundamental issues in structure-oriented programming environments is the separation of the internal program representation and the external views. Another interesting question is the interfacing between structure editing and text editing. The internal representation of HyperPro is a graph-like structure, and as such there are some similarities between HyperPro and programming environments based on graphs and graph grammars, such as Ipsen [13, 14]. However, in HyperPro significant units are edited as text. This is not the cased in environments like Ipsen, in which the structural entities are "atomic".

5 Conclusion.

In this paper we have characterized a subset of all 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 it is possible to define user interfaces on rich hypertexts, 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 decorate and aggregate 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, which 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) are presented together, but kept apart in the underlying hypertext. The literate view (see figure 3) is an example of an aggregated presentation.

HyperPro is an experimental system, a prototype, which we use to get practical experience with our ideas. As of now (spring 94), the most significant part of HyperPro is the interaction engine, as described in this paper. The storage level in HyperPro is realized by a HyperBase Fenris [4], a locally developed hyperbase and part of our ongoing line of hyperbase research (shortly described in [21]) 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] Robert M. Akscyn, Donald L. McCracken, and Elise A. Yoder. KMS: A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM*, July 1988.

[2] John Bentley. Programming pearls: Literate programming. *Communications of the ACM*, 29(5):364–369, May 1986.

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

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

[5] 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.

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

[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, volume 17, number 5*, pages 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] Frank Halasz and Mayer Schwartz. The dexter hypertext reference model. In *Proceedings of the NIST Hypertext Standardization Workshop, Gaitersburg MD, Jan 16-18*, 1990.

[11] 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.

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

[13] C. Lewerentz and M. Nagl. Incremental programming in the large: Syntax-aided specification editing, integration and maintenance. In Bruce D. Scriver, editor, *Eithteenth Hawaii International Conference on System Sciences*, pages 638–649, 1985.

[14] Claus Lewerentz. Extended programming in the large in a software development environment. In Peter Henderson, editor, *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 173–182, November 1988. *Software Engineering Notes*, Volume 13(5), November 1988 and *Sigplan Notices*, Volume 24(2), February 1989.

[15] 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*, 1991.

[16] 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*, 1991.

[17] J. Puttress and N. M. Guimaraes. The toolkit approach to hypermedia. In A. Rizk, N. Streitz, and J. André, editors, *Hypertext: Concepts, Systems and Applications*, pages 25–37, 1990.

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

[19] Johannes Sametinger and Gustav Pomberger. A hypertext system for literate c++ programming. *Journal of Object-Oriented Programming*, pages 24–29, January 1992.

[20] 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.

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