Towards an Option Space for Network Structures in Computer-Based Instruction

PAULA KOTZÉ
Department of Computer Science and Information Systems
University of South Africa, PO Box 392, Pretoria
E-Mail: kotzep@risc1.unisa.ac.za
Facsimile: (012)429-3434

Abstract: There are three major agents in a computer-based instruction (CBI) environment: the tutor, the tutoring system and the tutee. The emphasis of this paper is on the tutoring system which contains the instructional content and structures for presenting the content, the rules guiding a particular student's path through the material, and information regarding the performance data to be recorded. A CBI program consists of a network of related chunks of information, called *nodes*, that are connected by means associated *links*. Research has been undertaken into the general structures of a CBI environment and some of the results will be discussed in this paper.

The structure that makes up a CBI program is essentially a directed semantic multigraph. Formal specifications are employed to discuss several properties that influence the usability of these CBI networks structures, including reachability, determinism, precedence ordering and restartability. Although these structures are discussed in the context of CBI, the particular properties can also be transferred to other multimedia applications.

Keywords: Tutoring systems, computer-based instruction, human-computer interaction, usability, multimedia, graph theory, formal software specification.

Computing Review Category: H.5.1, C.4, K.3.1, D.2.1

1 Introduction

Formal approaches to software development (for example [2, 5]) have been mostly concerned with problem descriptions where the key aspects of the design state is functional, but avoid the expression of interactive behaviour. In the design of interactive systems it is also necessary to be concerned with the interaction between the system and its users. This concern is asserted by defining models of interactive behaviour where the issue is closer to what the user perceives. Operations are of interest in as far as they manipulate perceivable entities.

The design of interactive systems may be elucidated and enhanced by:

- Clarifying design dimensions relevant to the comprehension of interactive systems; and
- Providing criteria to expedite choice between design options [1].

An abstract system model of interactive behaviour can be employed to discuss the design of interactive systems, formulate interactive properties of various kinds, describe the relationship between task and system, as well as what happens when there is more than one independent party in the interaction. The aim of this paper is to show how such an abstract model of authoring

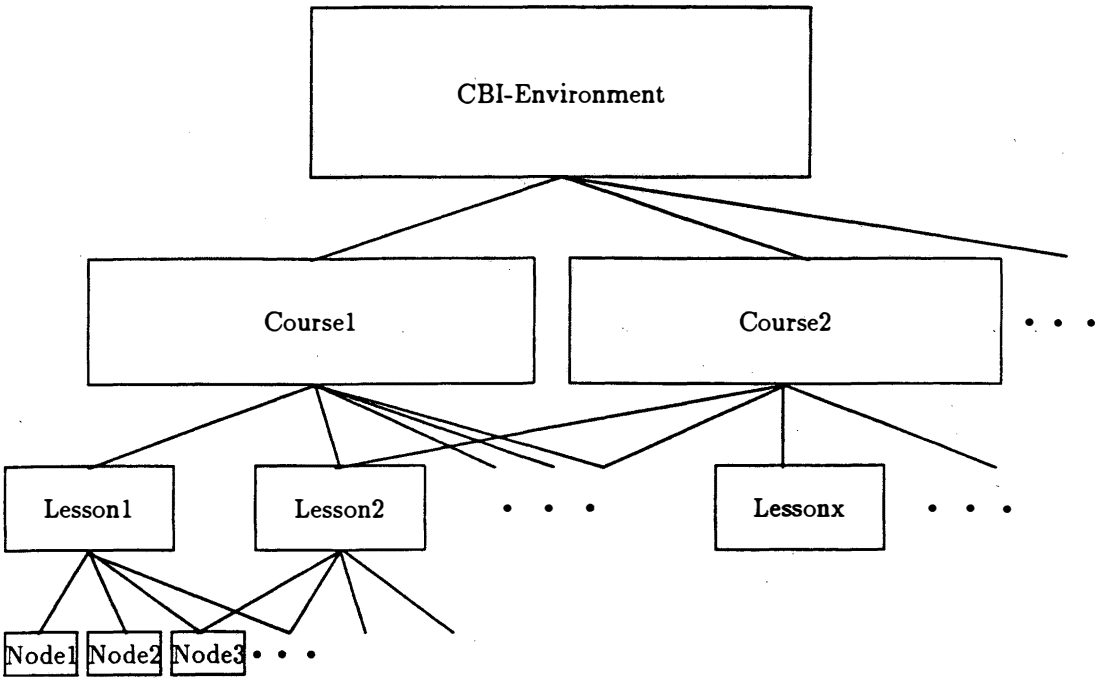

Figure 1: Computer-based instruction environment

support environments may be used to generate a precise framework or *option space* within which design choices may be discriminated.

A mathematical (formal) description of system behaviour provides the means to analyse aspects of the behaviour of a system such as usability and performance. A precise description of characteristics of the system can also be used as a basis for a set of generative user-engineering principles which can be used in the analysis, design and construction of future systems.

> *Authoring support environments (ASEs) are computer software tools which support authors in translating their subject knowledge and instructional needs into computer-based instructional material. These systems should provide functions to assist the author of such material in the execution of tasks forming an essential part of such material, for example, screen display, answer analysis, response judging, student data collection, control of multimedia devices, and feedback. It should also assist the author in creating the necessary data structures which constitute the instructional components, a computer-based instructional environment consists of.*

Figure 1 gives an illustration of a computer-based instruction environment. A computer-based instruction environment consists of a set of courses on a certain topic. Each course is subdivided into a number of lessons. Each lesson consists of a network of nodes connected by means of links. Nodes carry the actual material used for instruction. The functional characteristics of ASEs should reflect these levels of a CBI environment. Regardless of the kind of interaction an ASE involves, it should, as a minimum requirement, possess the general features required in the courseware development process of the three sublevels indicated [3].

The lowest level, that of node or instructional content creation, refers to the input, formatting and modification of nodes carrying media objects consisting of text, graphics, audio, video, or any other information which will be displayed or stored in relation to the nodes. This level is the most device-dependent since the way the content is entered or modified, and the type of presentation possible are all functions of the particular hardware involved. Further principle categories of instructions involves those related to the display or presentation of information, to student response processing, and those related to branching sequences. The nodes should be linked to form the networks which will be traversed by a student as a result of certain responses and rules. The creation, formatting and modification of such links, thus forms the second major function performed on this level.

The second level, that of lesson or instructional network definition, includes functions which involve specifying the structure of individual lessons. Lesson management functions include the capability to define or select a particular instructional strategy (test, tutorial, simulation, etc.), the specification of response data to be collected, the capability to test a lesson just created, and the control options available to a student. The effect of these functions is local to a specific lesson. For example most instructional systems automatically produce records of students whenever the student responds to a question, or passes a checkpoint (for example a particular node in a lesson, a certain percentage (performance) mark, or the end of a particular lesson). Such records typically include data such as the student's number (identification), a node identifier, time and date, total time spent on a node (response latency), the number of passes through a particular node (or attempts at a question), and the transcript of any response. A similar situation exists for student control options. A student may be allowed to go back one node, repeat a node, skip a node, restart a lesson, skip to the end of a lesson, etc. The capability to specify such control options, for the parts of lessons where it is permissable, thus also belongs to this level.

Global effects across lessons are specified at the course level. Fundamentally operations at this level allow an author to manage the authoring and instructional process for a whole course under development. The records accumulated for each lesson that forms part of a particular course are integrated and result in a very detailed trace of a student's progress through a particular course, which can be summarised and processed into reports as required. These records can be used to determine a student's access to other lessons in the course. The access controls for all lessons in a course are handled on this level, both for students and authors.

ASEs are used to produce computer-based instructional systems and usually have two areas of functionality :

1. Computer-based instruction management — organising networks into lessons, organising lessons into courses, keeping student records (assigned lessons, progress in assigned lessons, including score), run-control of lessons, etc.

2. Developing the networks that lessons are composed of — a lesson consists of a number of network structures connected in some or other way in order to achieve some instructional objective. Each network consists of chunks of information, called nodes, connected by means of links. ASEs supply the means to "program" these node-link structures.

In order to develop a (formal) model for evaluating ASEs we first need to establish the characteristics of the node, link, and network structures which are created by these environments. The different kinds, essential characteristics and control structures of each need to be identified.

The next step in developing the model would be to identify certain properties that these network structures should adhere to, for example,

- reachability (of nodes within a network, as well as the end of the network),
- directedness of the networks,
- determinism, etc.

The same needs to be done for the higher order structures — the lessons and the courses these networks would finally form part of. For lessons, for example, we need to model the scoring mechanisms, the control of prerequisites, the lesson status (whether it is still in the development phase, ready for full distribution or limited distribution, etc.). Courses form the highest level of structures identified and consists of a number of associated lessons (and therefore also the prerequisites associated with each lesson).

We also need to look at the different perspectives the two main groups of users of ASE, teachers and students, will have of the instructional system. What are the objectives of the two groups, how do they differ? Do the state and display of the system differ for the two groups? We need to establish the way in which the different roles are controlled by the ASE. The concept of initiative comes into play at this stage.

Research has been conducted and an extensive formal model constructed covering most of the aspects described above. The rest of this paper will report on some of the results as far as the networks structures are concerned.

2 Network definition

A CBI network consists of a number of nodes connected by means of a set of links. We will treat the set of nodes and the set of links as given sets for the purpose of this document. A complete description of these objects can be found in [4].

2.1 Given sets

The first set of objects we require is the set of nodes carrying the instructional content:

Nodes

$node\_id : NODE\_ID$
$Node\_Type$
$Node\_Contents$

Three typed partitions to a node type can be identified:

Node_Type

$node\_type : NODE\_TYPE$
$orientation : ORIENTATION$
$controlled\_by : CONTROLLED$

The contents of a node consists of two distinct but related sets of information — one contains the information "displayed" to a "student" and the other contains the "embedded monitor" information (node level controls).

Formally:

Node_Contents

$instructional\_contents : \text{seq}\ Media\_Objects$
$node\_monitors : \mathbb{P}\ MONITORS$

If one plans to create instructional and training material instead of more generalised interactive presentations, you need some special features that allow you to evaluate the students using the courseware. An instructional system should be able to keep records of the students participating and permit the extraction of such information from the material. The records might include information about the number of correct (or incorrect) answers a student has made, how long it took the student to finish a particular piece of work, the current score of the student, the scoring mechanism for any particular node, flags set to indicate how many times he has been to that node, or the path the student followed to get to the particular node, etc.

This information is not generally available to the student and he is usually unaware of the existence thereof, and we therefore say the information is "embedded" in the node contents. The node monitors are used for keeping student records as well as to determine which path to follow through the network representing the CBI material. The values of the node monitors thus change dynamically as result of a student's interaction with the instructional system. We will use node monitors in our discussion of a network type in section 7.3.

Record must also be kept of the nodes that actually exist:

Nodes_that_exist

$node\_instances : NODE\_ID \rightarrowtail Nodes$

$\forall\ nid : NODE\_ID \mid nid \in dom(node\_instances) \bullet$
$\quad (node\_instances(nid)).node\_id = nid$

Secondly, we need to define link objects, the glue that connects nodes into networks:

Links

$link\_id : LINK\_ID$
$Link\_Type$
$Link\_Contents$

Each link is associated with two nodes — a source node and a destination node, both of which should be existing nodes. A link can be sourced to one node only and has only one destination node. Several links can, however, be sourced to the same node, and more than one link may have both the same source and destination just with different conditions attached.

Each link could have a number of conditions associated with it — conditions that should be satisfied if a certain path is allowed to be taken (these conditions are usually related to the monitors associated with the nodes). Typical link conditions include the minimum score level required for traversing a particular path, the experience level of the student (novice, intermediate, advanced), how many times a student has visited the particular node, whether an answer to a question was correct or not, a particular choice a student might have made from a list, etc. Link conditions are fixed and do not change dynamically as is the case with the node monitors with which they are associated. The node monitors thus gather the data to be used by the link conditions in determining the path for traversal through the system.

Link_Contents

$$
\begin{aligned}
&link\_conditions : \mathbf{P}\ CONDITIONS \\
&link\_source : NODE\_ID \\
&link\_destination : NODE\_ID \\
&\Xi Nodes\_that\_exist \\
&\hline
&link\_source \in \mathrm{dom}(node\_instances) \\
&link\_destination \in \mathrm{dom}(node\_instances)
\end{aligned}
$$

Link types convey a certain amount of information and allow one to distinguish among different relationships between nodes, and can thus be compared with links in a *semantic network*. A distinction is made between those link types used for ***control flow*** and those used as ***data flow***. Control flow specifies the sequence of actions to solve a given problem, while data flow defines the flow of information without determining an order of processing this information.

Link_Kind

$$
\begin{aligned}
&link\_type : LINK\_TYPE \\
&attachment : ATTACHMENT \\
&used\_for : USE
\end{aligned}
$$

We also have to keep record of all the links that exists in our CBI system:

Links_that_exist

$$
\begin{aligned}
&link\_instances : LINK\_ID \rightarrowtail Links \\
&\hline
&\forall\ lid : LINK\_ID \mid lid \in \mathrm{dom}(link\_instances) \\
&\quad (link\_instances(lid)).link\_id = lid
\end{aligned}
$$

Given these two sets of objects, *node_instances* and ***link_instances***, we can define a CBI network.

2.2 General network definition

Each network is identified by means of a *network_id*, chosen from a representative set *NETWORK_ID*. Each network has a type and a contents, carrying the node-link associations that make up the network.

Networks

$$
\begin{aligned}
&network\_id : NETWORK\_ID \\
&Network\_Type \\
&Network\_Contents
\end{aligned}
$$

3 Network contents

Each CBI network has a root node which indicates the start of the network ("Start" in Figure 2) and an indication of when the end of the network has been reached ("End" in Figure 2).

The graph constituting a CBI network can be defined in a similar way to that of a general graph as consisting of a non-empty set N representing the finite set of nodes that form part of the network, and a finite set L which forms the edges (links) of the graph, and a mapping ϕ from the set of edges L to the set of pairs of elements of N.

The mapping ϕ is defined by determining the transitive closure[1] of the root node. The partial function

$$destination\_of : NODE\_ID \pfun \mathbb{P}\, NODE\_ID$$

defines the set of nodes that can be reached from any given node. The function takes a single node reference as argument and returns a set of nodes that are direct destinations of links which have their source in the argument node.

The partial function

$$links\_attached : NODE\_ID \pfun \mathbb{P}\, LINK\_ID$$

defines the set of links that can be reached from any given node. The function takes a single node reference as argument and returns a set of links that have their source in the argument node.

Collectively:

Destination_Nodes

$$destination\_of : NODE\_ID \pfun \mathbb{P}\, NODE\_ID$$
$$links\_attached : NODE\_ID \pfun \mathbb{P}\, LINK\_ID$$
$$\Xi Nodes\_that\_exist$$
$$\Xi Links\_that\_exist$$

$$\forall nid : NODE\_ID \mid nid \in \mathrm{dom}(node\_instances) \bullet$$
$$(destination\_of(nid) =$$
$$\{lid : LINK\_ID \mid nid = (link\_instances(lid)).links\_source \bullet$$
$$(link\_instances(lid)).link\_destination\} \land$$
$$links\_attached(nid) =$$
$$\{lid : LINK\_ID \mid nid = ((link\_instances(lid)).link\_source) \bullet$$
$$((link\_instances(lid)).link\_id)\})$$

In order to come up with the set of all nodes existing in a particular network we need to calculate the transitive closure of the root node.

The partial function

$$reachable\_nodes : NODE\_ID \pfun \mathbb{P}\, NODE\_ID$$

calculates the transitive closure of a node in the context of the whole network, i.e. the function takes any single node in the network as its argument and returns the set of all nodes that can be reached from that node. This is done by firstly including the set of nodes that can be reached directly from a certain node, and then add to this the set containing all the nodes that can be reached from these nodes.

The partial function

$$reachable\_links : NODE\_ID \pfun \mathbb{P}\, LINK\_ID$$

calculates the set of all links existing in a particular network in a similar way.

[1] Transitive closure - the set of vertices (nodes) that can be reached from a given vertice (node) by traversing edges (links) from the graph in the indicated direction[6].

All_Reachable_Nodes

$$
\begin{array}{l}
reachable\_nodes : NODE\_ID \pfun \mathbb{P}\, NODE\_ID \\
reachable\_links : NODE\_ID \pfun \mathbb{P}\, LINK\_ID \\
\Xi Destination\_Nodes \\
\hline
\forall\, nid : NODE\_ID \mid nid \in \mathrm{dom}(node\_instances) \bullet \\
\quad (destination\_of(nid) = \varnothing \Rightarrow \\
\qquad reachable\_nodes(nid) = \varnothing) \land \\
\quad (destination\_of(nid) \neq \varnothing \Rightarrow \\
\qquad (reachable\_nodes(nid) = destination\_of(nid) \cup (\bigcup \{rnid : NODE\_ID \mid \\
\qquad\quad rnid \in destination\_of(nid) \bullet reachable\_nodes(rnid)\})) \land \\
\qquad (reachable\_links(nid) = links\_attached(nid) \cup (\bigcup \{rnid : NODE\_ID \mid \\
\qquad\quad rnid \in destination\_of(nid) \bullet reachable\_links(rnid)\}))
\end{array}
$$

We are now in the position to define the general contents of a CBI network. Network contents consists of the following:

- The identifier of the root node indicating the start of the network.
- The set of nodes that make up the network.
- The set of links that connect the nodes to for an instructional network.

To define the set of nodes that make up a network, we need to compute the transitive closure of the root node in the context of the whole network environment.

$$nodes\_in\_network = reachable\_nodes(root\_node) \cup root\_node$$

To define the set of links that make up a network, we need to compute the transitive closure of the root node in respect of the links originating from it.

$$links\_in\_network = reachable\_links(root\_node)$$

Integrity checks to perform include determining that all the nodes and links used in defining the node contents actually exist, and that all the links used in the network is sourced to a node in that network.

We can therefore define the contents of a network collectively as follows:

Network_Contents

$$
\begin{array}{l}
root\_node : NODE\_ID \\
nodes\_in\_network : \mathbb{P}\, NODE\_ID \\
links\_in\_network : \mathbb{P}\, LINK\_ID \\
\Xi All\_Reachable\_Nodes \\
\hline
root\_node \in nodes\_in\_network \\
nodes\_in\_network \subseteq \mathrm{dom}(node\_instances) \\
nodes\_in\_network = reachable\_nodes(root\_node) \cup \{root\_node\} \\
links\_in\_network \subseteq \mathrm{dom}(link\_instances) \\
links\_in\_network = reachable\_links(root\_node) \\
\forall\, lid : LINK\_ID \mid lid \in links\_in\_network \bullet \\
\quad (link\_instances(lid)).link\_source \in nodes\_in\_network
\end{array}
$$

4 Network types

As is the case with components making up networks, i.e. nodes and links, networks also belong to a particular type. The type of a network is determined by the behaviour of the network.

The general network type can be defined in terms of five subtypes identified, each representing a different network behaviour:

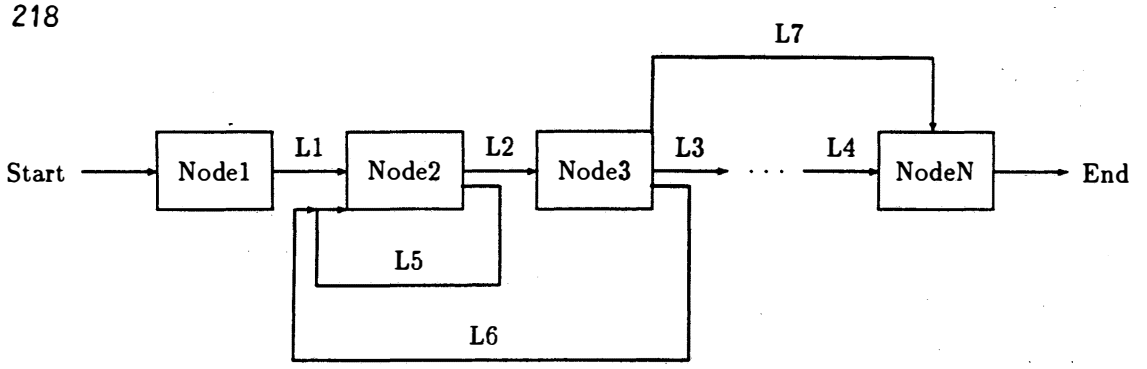

Figure 2: Non-linear network structure

$$Network\_Type \mathrel{\widehat{=}} Linear\_Network \lor Non\_Linear\_Network \lor Contextual\_Network \lor Browsing\_Network \lor Detour\_Network$$

Thus we have:

Network_Type

$network\_type : Network\_Type$

One of these subtypes, the contextual network, and its behaviour will be discussed in the following sections to illustrate the concept of network behaviour.

5 General network behaviour

Network behaviour can be specified in terms of the path followed through the network by a (student) user. This is done in terms of a current node, a predecessor node, a successor node and a history of the path already taken.

Network types can to some extent be determined by the types of the nodes that are allowed to be part of the network. A browsing network could for example only contain nodes of type *browsing*, while a detour network can only contain nodes of type *glossary*, *information*, or external node of an unknown type. Sometimes no restrictions are placed on the node types than may be included.

Network_Behaviour

$\Xi Network\_Contents$
$current\_node : NODE\_ID$
$predecessor\_node : NODE\_ID$
$successor\_node : NODE\_ID$
$history : seq NODE\_ID$
$types\_of\_nodes : \mathbb{P}\, NODE\_TYPE$

$root\_node = last(history)$
$current\_node \in nodes\_in\_network$
$\mathrm{ran}(history) = \mathrm{dom}(node\_instances)$
$predecessor\_node = head(history)$

Several general network properties can be defined, of which three are briefly discussed below.

5.1 Reachability

If the end of a network can be reached from any given node in the network, the network is said to adhere to the property of *end_reachable* if and only if, there exists at least one node in the set of nodes reachable from that node, which has the *end* link as its destination.

End_Reachability

$end\_reachable : \mathbb{P}\ NODE\_ID$
$\Xi Network\_Contents$
$\Xi Link\_type\_sets$

$\forall nid : NODE\_ID \mid nid \in nodes\_in\_network \bullet$
$nid \in end\_reachable \Leftrightarrow$
$\exists rlid : reachable\_links(nid) \bullet$
$link\_type\_of(rlid) = end$

5.2 Determinism

We are now in the position to define a deterministic and a non-deterministic network. A deterministic network is defined as a network where the end of the network can be reached from all of the destination nodes attached via links to any particular node in that network — i.e. a user will never be trapped in the network without being able to reach the proper end of the network.

Deterministic_Network

$deterministic : \mathbb{P}\ NODE\_ID$
$\Xi End\_Reachability$

$\forall nid : NODE\_ID \mid nid \in end\_reachable \bullet$
$nid \in deterministic \Leftrightarrow$
$\forall rnid : reachable\_nodes(nid) \bullet$
$\exists rlid : reachable\_links(rnid) \bullet$
$link\_type\_of(rlid) = end$

A non-deterministic network, on the other hand allows for the case where at least one of the paths would never reach the end, and thus permits for a user to be trapped in some or other circular route, for example.

Non_Deterministic_Network

$non\_deterministic : \mathbb{P}\ NODE\_ID$
$\Xi End\_Reachability$

$\exists nid : NODE\_ID \mid nid \in end\_reachable \bullet$
$nid \in non\_deterministic \Leftrightarrow$
$(\exists rnid : reachable\_nodes(nid) \bullet$
$end \notin link\_set\_types\_of(reachable\_links(rnid))$

Although a CBI network should essentially be deterministic (but not necessarily predictable), it may also allow for non-determinism. For example, in the cases where unlimited revision of a piece of material is allowed.

5.3 Directionality

The order in which nodes appear in a network is important. For example in a tutorial, a question node cannot appear before the node containing the appropriate content material has been displayed.

Both linear and non-linear networks are found in CBI systems. Linear networks are incorporated in systems which calls for a linear progression through a piece of material, for example in the case of a test or a questionnaire. For the purpose of this document we will, however focus on non-linear networks. Two different ordering operators have been defined to represent the relationship amongst nodes in linear and non-linear networks. The operator for non-linear networks are represented by:

$[X]$

$\_ \preceq \_ : X \leftrightarrow X$

$\forall x, y, z : X \bullet$
$x \preceq x \vee$
$(x \preceq y \wedge y \preceq z \Rightarrow x \preceq z)$

This ordering operator allows for reflexivity and transitivity, but not for symmetry or antisymmetry. For full details about the derivation of this operator, see [4].

6 Non-linear networks

Non-linear networks are networks that can behave as illustrated in Figure 2. It allows the return to nodes already traversed through, or in other words, the same node can precede itself, or both precede or succeed any other node in the network. Loops are allowed in a non-linear network.

6.1 Successor nodes

Each node in a non-linear network can be associated with any number of associated successor nodes, or none at all in the case of the last node in the network:

$$\forall\, nid : NODE\_ID \mid nid \in nodes\_in\_network \bullet \#(destination\_of(nid)) \geq 0$$

Both forwards and backwards progression is allowed. The *successor_node* can therefore be a member of the sequence of history nodes. The *predecessor_node* is the last element added to the history sequence. The successor node can be chosen from a set compiled by obtaining the union of the following three subsets:

- The set of history nodes.
- The current node.
- The destination node of any other link attached to the current node.

Thus,

$$\begin{aligned}&\forall\, nid : NODE\_ID \mid nid \in nodes\_in\_network \bullet \\ &\quad (successor\_node \in destination\_of(nid) \land \mathrm{ran}(destination\_of) \subseteq \\ &\qquad \{\mathrm{ran}(history) \cup \{current\_node\} \cup \{(link\_instances(lid).link\_destination)\}\}\end{aligned}$$

where $lid \in links\_attached(nid)$.

6.2 Non-Linear Directionality

Looking at the precedence order between the *current_node* in a non-linear network and the nodes in the history we can observe that:

$$\begin{aligned}&\forall\, cnid : NODE\_ID \mid cnid = current\_node \bullet \\ &\quad (\forall\, hnid : NODE\_ID \mid hnid \in \mathrm{ran}(history) \bullet cnid \succeq hnid \lor \\ &\quad \exists\, phnid : NODE\_ID \mid phnid \in \mathrm{ran}(history) \bullet cnid \preceq phnid) \land \\ &\quad \forall\, snid : NODE\_ID \mid snid \in (nodes\_in\_network \setminus (\mathrm{ran}(history))) \bullet cnid \preceq snid\end{aligned}$$

6.3 Non-linear network type

We can thus define the general non-linear network type as:

Non_Linear_Network

Δ*Network_Behaviour*

$$
\begin{array}{l}
** \text{ Types of nodes allowed in network } ** \\
\{browsing\} \notin types\_of\_nodes \\
** \text{ Successor nodes and history } ** \\
\forall nid : NODE\_ID \mid nid \in nodes\_in\_network \bullet \#(destination\_of(nid)) \geq 0 \land \\
\quad (\forall lid : LINK\_ID \mid lid \in links\_attached(nid) \bullet \\
\qquad (link\_instances(lid)).link\_destination \in nodes\_in\_network \lor \\
\qquad link\_type\_of(lid) = end) \land \\
\quad (successor\_node \in destination\_of(nid) \land \\
\quad \text{ran}(destination\_of) \subseteq \\
\qquad \{\text{ran}(history) \cup \{current\_node\} \cup \{(link\_instances(lid).link\_destination)\}\} \\
** \text{ Directionality between current node and nodes in history, } ** \\
** \text{ and current node and nodes not in history } ** \\
\forall cnid : NODE\_ID \mid cnid = current\_node \bullet \\
\quad (\forall hnid : NODE\_ID \mid hnid \in \text{ran}(history) \bullet cnid \succeq hnid \lor \\
\qquad (\exists phnid : NODE\_ID \mid phnid \in \text{ran}(history) \bullet cnid \preceq phnid)) \land \\
\quad \forall snid : NODE\_ID \mid snid \in (nodes\_in\_network \setminus (\text{ran}(history))) \bullet cnid \preceq snid
\end{array}
$$

6.4 Node monitors

In this general non-linear network type no mention has been made to node monitors. The reason for that is that one can distinguish between a general non-linear network without the ability to monitor a student's progress through the system, and a network which do incorporate node monitors and adapt network traversal accordingly. The former is used for lessons with fixed routes (although allowing for some flexibility by the choice of nodes), while the latter, which we will call a *Contextual_Network*, allows for different routes through the same set of nodes for different performance levels. In both these types of networks, the choice of successor node is initiated by the system. We say the traversal is system controlled or determined.

The only type of node monitor sometimes associated with a general non-linear network is scoring, if applicable. Link conditions associated with such monitors are usually absent though.

7 Contextual networks

A contextual network is a non-linear network which makes extensive use of node monitors and associated link conditions.

7.1 Successor nodes

The choice of successor node is, as with a general non-linear network, restricted to a node from the set compiled by obtaining the union of all the history nodes, the current node, and the new nodes for which links exist in the current node. The choice is dependent on the directionality properties of the network.

7.2 Node monitors and link conditions

Several monitors are generally attached to nodes associated with these kind of networks. Associated links contain conditions based on the monitors which have to be satisfied for that path to be followed.

The association between the monitors and the conditions is represented by a function, *satisfies*, which maps the relations between them to a Boolean value (either true or false):

$$satisfies : (MONITORS \times CONDITIONS) \rightarrowtail Boolean$$

A certain path is followed if and only if all the conditions associated with the link is satisfied:

$$\forall l\_con : CONDITIONS \mid l\_con \in (link\_instances(lid)).link\_conditions \bullet$$
$$\exists n\_mon : MONITORS \mid$$
$$n\_mon \in (node\_instances(current\_node)).node\_monitors \bullet$$
$$satisfies(n\_mon, l\_con) = True$$

Conditions might for example be associated with the current score, learning processes and ability (different paths (strategies) may be followed for low, average and high ability), certain flags set by the nodes already traversed, etc.

7.3 Contextual network type

The contextual network type is thus a general non-linear network with additional properties related to node monitors and link conditions:

$$Boolean ::= True \mid False$$

Contextual_Network

$$\Delta Non\_Linear\_Network$$
$$satisfies : (MONITORS \times CONDITIONS) \pfun Boolean$$

** Types of nodes allowed in network **
$$\{question\} \in types\_of\_nodes$$
** Successor nodes **
$$\forall lid : LINK\_ID \mid lid \in links\_attached(current\_node) \bullet$$
$$successor\_node = (link\_instances(lid)).link\_destination \Leftrightarrow$$
$$\forall l\_con : CONDITIONS \mid l\_con \in (link\_instances(lid)).link\_conditions \bullet$$
$$\exists n\_mon : MONITORS \mid$$
$$n\_mon \in (node\_instances(current\_node)).node\_monitors \bullet$$
$$satisfies(n\_mon, l\_con) = True$$

8 Using such models

The question "so what?" can now be posed. What do we use such models for? To conclude this paper we will name but a few of these uses.

Using the networks structures as defined above, we can in the first instance define the different types of lessons that can be found in a CBI system.

A tutorial lesson, for example, allows for either a linear or non-linear progression through the associated networks, depending on the content material. This category of lessons include tutorials depending on student models (the so-called intelligent tutoring systems) or those that are not, since all of them have the same basic structure. The only difference is that the network of an intelligent tutoring system is always contextual. Detour networks including help and glossaries form integral parts of tutorials. Restart behaviour can be any of three types identified — restarting at the last active node (current node), restarting at the root node, or non-interruptable [4].

We can therefore identify an "ordinary" tutorial as:

$$Tutorial \mathrel{\widehat=} ((Non\_Linear\_Network \lor Contextual\_Network) \land Detour\_Network) \land$$
$$Deterministic\_Network \land$$
$$(Current\_Restart\_Behaviour \lor Root\_Restart\_Behaviour$$
$$\lor No\_Restart\_Behaviour)$$

or an "intelligent" tutorial as:

$$Intelligent\_Tutorial \mathrel{\widehat=} (Contextual\_Network \land Detour\_Network) \land$$
$$Deterministic\_Network \land$$
$$(Current\_Restart\_Behaviour \lor Root\_Restart\_Behaviour$$
$$\lor No\_Restart\_Behaviour)$$

Secondly, we can define a set of generic operations than can be performed on the different substructures identified. Such a comprehensive model of abstract data types and operations, based on sound principles and structures, can in refined form be used as a reference model for the development of new ASEs.

Furthermore the model allows one to reason about the interactive properties of different classes of ASEs — a classification that has been elucidated by means of the model. The model has been used to classify ASEs into three distinct categories according their user interface styles — map-based, display-based and code-based [4]. The model also allows one to reason and make precise observations about different properties that are not possible by informal argument.

Furthermore, such a model can be used as an evaluation framework against which existing ASEs can be assessed as far as the level of support they give to the other two agents in constructing and using CBI systems, is concerned.

Acknowledgement

I am grateful to my supervisor, professor Michael Harrison of the Human-Computer Interaction Group at the University of York, for his guidance and support since 1992. The research reported on in this paper, as well as my visits to York during 1992 and 1994, have financially been supported by means a prestige scholarship from the Centre for Science Development of the HSRC, as well as a grant from the Research and Bursaries Committee of the University of South Africa.

References

[1] Harrison M.D. 1992. A model for the option space of interactive systems. In: *Engineering for Human-Computer Interaction*, edited by J. Larson & c. Unger, Elsevier Science Publishers B.V. (North-Holland), p. 155-170.

[2] Jones C.B. 1990. *Systematic software development using VDM*. London: Prentice-Hall.

[3] Kearsley G. 1982. Authoring systems in computer based education. *Communications of the ACM*, 25(7):429-437.

[4] Kotzé P. 1995. Agents participating in a computer-based environment, Second Year DPhil Report, Department of Computer Science, University of York.

[5] Morgan C. 1990. *Programming from specifications*. Hemel Hempstead: Prentice-Hall International.

[6] Sedgewick R. 1988. *Algorithms*. Addison-Wesley.