Applying Patterns to Hypermedia Instructional Design (APHID)

A Thesis Submitted to the College of

Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

in the

Department of Computer Science

University of Saskatchewan

Saskatoon, SK

By

Judi R. Thomson

March 2000

Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and the University of Saskatchewan in any scholarly works which may be made of any material in my thesis.

Request for permission to copy or to make other use of material in this thesis, in whole or part, should be addressed to:

Head of the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan

Abstract

This research addresses the issue of automatically generating instructional hypermedia documents (in the form of web sites). Our hypothesis is that, for certain types of hypermedia, an automated approach can produce satisfactory hypermedia applications more efficiently than humans are able to create them. We propose a method (APHID) that guides a hypermedia creator through the design process and partially automates the creation of hypermedia applications. Our method uses concept maps and instructional design patterns, as well as the more common domain and presentation models, to support partial automation for creating instructional hypermedia.

Most hypermedia application developers follow basic graphical design principles, but few commonly accepted principles exist for the structuring of hypermedia applications. The design of instructional hypermedia imposes the additional requirement that the designer be expert both in hypermedia design and in instructional design. APHID supports designers through the use of patterns to describe and clarify design concepts for both instructional design and interface design.

This thesis describes the design and development of the APHID approach and a prototype software tool that supports the development of instructional hypermedia using the APHID approach. The hypermedia generated by APHID can be used by instructors in a textbook-like fashion. The thesis also presents a study in which web sites created with APHID are compared (by an independent evaluator) to web sites created by instructional technologists. The study shows that good instructional web sites can be generated semi-automatically with less expenditure of time on the part of the instructional designer.

Acknowledgments

I'd like to thank the faculty and staff of the Department of Computer Science at the University of Saskatchewan. The openness and helpfulness of the people in the department make it one of the best place I can imagine to do graduate work. My research was funded in part by the University of Saskatchewan, in part by the Natural Sciences and Engineering Research Council (NSERC) and in part by Saskatchewan Education. I am grateful to all three organizations for their support. I would also like to thank Paul Sorenson and Eleni Stroulia for their support over the past year.

Over the past few years, my supervisors, John Cooke and Jim Greer, have advised, calmed, supported, listened, structured, read and edited- all without complaint! Thank you both. I also acknowledge the members of my thesis committee, who provided excellent guidance. Thanks go as well to Dr. Peter Brusilovsky, who kindly agreed to be the external examiner and who gave well placed and constructive advice at more than one point in my research.

Several friends helped out with this research in one way or another. Laurissa Tokarchuk, Guus Van de Velde, everyone who participated in the study and the 371 students who calculated website metrics deserve specific mention. Special thanks to Tim and to Lori- you were both there whenever I needed you to be.

Finally, I must thank my family. My parents have been supportive throughout the adventure. My children, Kirsten and DJ, displayed patience well beyond what could be expected. And Jamie-this would not have been possible without your love and support. Thank you all!

Contents

Figures

Tables

Chapter One

Introduction

1.1 Information and the World Wide Web

Given the immense popularity of the web and its potential usefulness as an information source, it is surprising that it remains one of the most under-designed global sources of information available. Information on the web is difficult to find, hard to authenticate, and tedious to produce. The potential of the web as a resource for instructional use is tremendous. Information about any topic is usually freely available and accessible using basic software. Unfortunately, the poor design of most of the web renders it difficult to use in any real instructional setting. Even though the World Wide Web is one of the most readily accessible sources of information, because of poor design and usability it remains underutilized by learners; and it is learners who would most benefit from freely accessible information.

Unfortunately, the poor design of web applications affects learners even more than other kinds of users. When a Web-based hypermedia application is intended to be instructional, the quality and maintainability are even more important because the end users are usually unsophisticated. Instructional systems that are poorly designed often fail to provide any real instruction to the intended learners. Instead learners become frustrated and confused by the choices presented. Often they form no solid understanding of the concepts the instructional site is attempting to communicate.

The problem of poor design is exacerbated when the developers of hypermedia applications are not software professionals. Frequently, hypermedia application design has become the task of graphic artists or content area experts, such as instructors. Software professionals are called in when interactivity and scripting are required, but people who are untrained in software design often do the overall application design.

Novice web developers need guidance to produce well-designed hypermedia and that guidance can be partially provided through a well-defined development process. A flexible development process that can be used for a wide range of hypermedia applications is needed.

Most hypermedia developers follow basic graphical design principles (Nielsen 1993), but few commonly accepted principles exist for structuring and designing the navigational behaviour of the hypermedia. Design processes for hypermedia applications do exist (Isakowitz, et al. 1995), but are often tailored for use with applications having one particular purpose. When the process is applied to an application with a different purpose, it often no longer works effectively. Instead an ad-hoc application design process emerges with little control for quality or maintainability. The resulting hypermedia application may be quite useable within a narrow set of purposes, but is often difficult to scale, hard to adapt for different uses and extremely time consuming to maintain.

Typically we think of the purpose for instruction as simply to learn about a particular topic. While the overall goal of instruction is to communicate the concepts within a certain domain, it also always has a secondary purpose. For example, an instructor might wish to provide an overview, a review, or a detailed presentation. Goals such as those are the secondary purposes for instruction. The sequence in which an instructor presents information depends both on the purpose for instruction and on the underlying structure of the concepts with which the instruction is concerned. Web sites that are intended to be instructional must reflect these different instructional purposes, as well as the structure of the actual material in their design and presentation.

A design process for instructional hypermedia applications must consider both the traditional approaches for designing hypermedia applications as well as the instructional purpose and the underlying conceptual structure of the instructional domain. This research investigates the incorporation of a concept map into the design process for instructional hypermedia to facilitate the representation of both domain structure and instructional purpose. The resulting process considers traditional design issues such as domain modelling and user interface as well as issues of instructional design.

1.2 Research Goal

The goal of this research is to define an improved, partially automated process for the development of instructional hypermedia applications. For the purposes of this research, instructional hypermedia applications are defined to be any hypermedia application that is intended to 'teach' one or more concepts. Often, but not exclusively, instructional applications will be implemented as tutorials. When an application is instructional, there is an implied 'sequence' to the concepts in the instructional domain and while it may be quite loosely defined, it represents one optimal path through the conceptual space. Development processes for general hypermedia applications usually do not consider the paths through the material as part of the design process. This difference between instructional hypermedia and other hypermedia renders existing development methods inadequate.

Instructional hypermedia has (or should have) an instructional purpose or instructional goal. Typically the goal is that the learner should have gained some knowledge after working with an instructional application. One measurement of how well learners understand material is to determine their ability to reconstruct the concept map. Instructional design is concerned with deciding how best to present the concepts to give the learner the desired understanding of the conceptual space. Within a hypermedia application, the instructional design is accomplished, in part, by carefully planning the learner's path through the material. The desired sequence is determined by the instructional purpose and by the structure of the concept map for the instructional domain. Existing hypermedia development processes have no mechanism for representing this type of conceptual information in the application design.

Existing hypermedia design methods are primarily concerned with user interface design and data modelling. A design method for instructional systems must also be concerned with the structure associated with the concepts in the domain of instruction and with representing the possible instructional strategies as part of the models. This research has defined such a process.

1.3 Analysis and Design of Instructional Hypermedia

An educational hypermedia application is any hypermedia application intended to provide new knowledge to the user. In that light, almost every web site could be considered an educational application. In contrast, an instructional hypermedia application is one that provides a definite sequence of instructional activities with the intent of imparting a specific set of knowledge to the user. Instructional hypermedia applications are a subset of educational applications. The design of instructional hypermedia is a process similar to the design of any hypermedia application with the added requirements of providing a path through the hypermedia appropriate for achieving the instructional goal.

Most hypermedia design methods concentrate on creating static models of the hypermedia application. Other than data modelling, the major considerations during design are presentation-layer decisions, such as the placement of images and text, perhaps with some modelling of event handling at the user interface level. The organization of the data is considered during the design of an application, but the semantics of that data are not. While the organization of the data, and the relationships between data elements, do influence the structure and navigability of a web site, they do not give enough information to make suggestions about sequencing. In order to make decisions about the sequencing of information one needs knowledge about the semantics of the data. To improve the design of instructional hypermedia applications, guidelines for determining suitable sequences through the material are necessary.

1.3.1 Models for Hypermedia Analysis and Design

A hypermedia application is simply a set of individual documents linked together via hyperlinks. The documents within a hypermedia application have some united theme, such as a site for a University department or a site for a course. Also, the documents within an application usually share common user interface elements and have a similar design. In short, a hypermedia application is a cohesive set of individual hypermedia documents.

Typically the design process for a hypermedia application involves first selecting a topic, then creating a data model of the domain so that the static structure of the information is understood, and finally creating a model of the interface and user elements that will be a part of the application. The resulting application provides navigation options for the user based on the structure of the data. The assumption made is that the bulk of the information within the application is well structured with identifiable attributes.

Data modelling is a valuable method of gaining an understanding of the structure of the information within the instructional domain. From the data model one can determine the relationships among the individual data elements within the instructional domain. These relationships can be used to provide some of the navigation opportunities within the application such as allowing users to navigate from descriptions of classes to the home page of the instructor for the class (using the relationship classes *have* instructors). Indeed, for casual users who wish to gain a general understanding of the domain, data model navigation may be the only type of navigation required.

When the purpose of the site is to provide direct instruction on a particular topic or a particular set of skills, another kind of navigation is required. Learners are most successful in learning a specific set of skills or facts when concepts are presented in an organized and controlled fashion (Oliver 1995). In most traditional instructional settings (i.e. classrooms), the instructor sets the sequence through a given set of material. An experienced instructor is skilled at determining the best sequence for a particular learner and communicating that sequence to the learner. Instructors choose the sequence based on the static structure of the data, the semantics of the individual data elements and the needs of the learners.

Simple data modelling does not consider the semantics of the data in an instructional sense. It provides a general picture of the kind of data that makes up a domain, but does not provide an understanding of how the domain is constructed semantically. One way of representing the semantic model of a domain is through a concept map. A concept map represents the ideas in a domain as a graph, where the concepts are nodes in the graph and the relationships between concepts are arcs.

1.3.2 Patterns

Patterns are used by the software community as a concise method of describing the design (or analysis) of software. Specifically, patterns describe known solutions to common or recurring problems within the domain of software design. Their purpose is to facilitate accurate communication about design issues between software professionals. Catalogs of patterns are freely available for designers to use (Appleton 1997).

The principles of patterns can be applied to the construction of instructional hypermedia applications. Since instructors are not often software designers, and software architects are not often instructional experts, a language to facilitate communicating about hypermedia design is useful. Patterns can be developed and used to describe the design of instructional hypermedia applications. Rules to govern the sequence of instruction can be described in terms of patterns. Each pattern, in this case, would represent a known 'good' solution to the problem of selecting the next important piece of information for a particular user, given a particular instructional purpose. The pattern is a set of guidelines for choosing the 'next' concept from the concept map. The pattern captures both an instructional idea and a method for expressing that idea in a hypermedia application. In this research the principles of design patterns and object-oriented development have been applied to the construction of instructional hypermedia applications. The resulting process, APHID (Applying Patterns to Hypermedia Instructional Design) enables a wide range of developers to apply known good solutions to hypermedia design problems.

1.4 The APHID System

APHID is both a process for creating instructional hypermedia and a software tool that enables the creation of instructional hypermedia. Instructional designers begin by creating models of the intended instructional application which are then used to automatically generate the final hypermedia applications. APHID supports several different types of instruction (e.g., reviews, introductions) and helps designers to build hypermedia applications that are designed specifically for particular types of instruction.

The instructional strategies and interface components used by APHID are described using patterns. The use of patterns helps to ensure that both instructors and software practitioners have the same understanding of what effects specific features of the process have on the final hypermedia applications.

1.5 Research Objectives

The generation of instructional hypermedia requires both an understanding of the underlying concept structure for the domain in question as well as an understanding of design principles for instructional hypermedia applications. The main objective for this research is as follows:

> *To demonstrate that the APHID approach to designing and producing instructional hypermedia can produce satisfactory hypermedia applications more efficiently than humans are able to.*

To realize this objective several more general goals have to be met. These are:

1. To identify common components within a hypermedia application intended for instructional use. These components will be comprised of element types and relationship types. Any instructional application is composed of elements and relationships that are drawn from this set of common components.
2. To identify the current 'best practices' for the creation of instructional hypermedia applications and to codify those practices in the form of patterns. Some patterns are design patterns, while others are instructional patterns through the conceptual space. Each instructional pattern corresponds to a specific instructional strategy or goal.
3. To identify a development approach for hypermedia that utilizes patterns and concept maps to produce well designed instructional hypermedia applications. The production of the applications is partially automated.
4. To implement a prototype software system that embodies the APHID approach.
5. To evaluate the implemented system to demonstrate that the research objective has been met.

1.6 Outcomes

This research has resulted in a support system for the construction of instructional hypermedia applications called APHID (Applying Patterns to Hypermedia Instructional Design). The tool provides support for creating concept maps and for creating the data elements that instantiate the concepts in the map. The concept maps are constructed to conform with specifications developed as part of this research.

Once concept map and data elements are identified, the tool assists the educator to build a Web-based instructional hypermedia application from the data elements. It considers instructional purpose and strategy as well as learner ability when constructing the application. The application is generated automatically using instructional patterns to govern the selection of concepts to be presented.

1.7 Organization of the Thesis

This research draws from a number of areas in computer science including software engineering and hypermedia design. It also draws from the educational discipline of instructional design. Figure 1-1 illustrates the scope of this project. The outer star represents distinct research areas relevant to the project. The middle rings represent research areas that span across other research disciplines. For instance, metrics and patterns research is carried on within all four design disciplines. APHID draws from the metrics and patterns research in all four areas. As can be seen from the diagram, the research is broad and cannot expect to make major contributions in all areas. The main contributions of this work are in the area of instructional hypermedia design and implementation (noted in bold on the diagram).

Chapter two presents a survey of the relevant literature, concentrating on the areas with the most applicability to the research. Chapter three contains the analysis required to

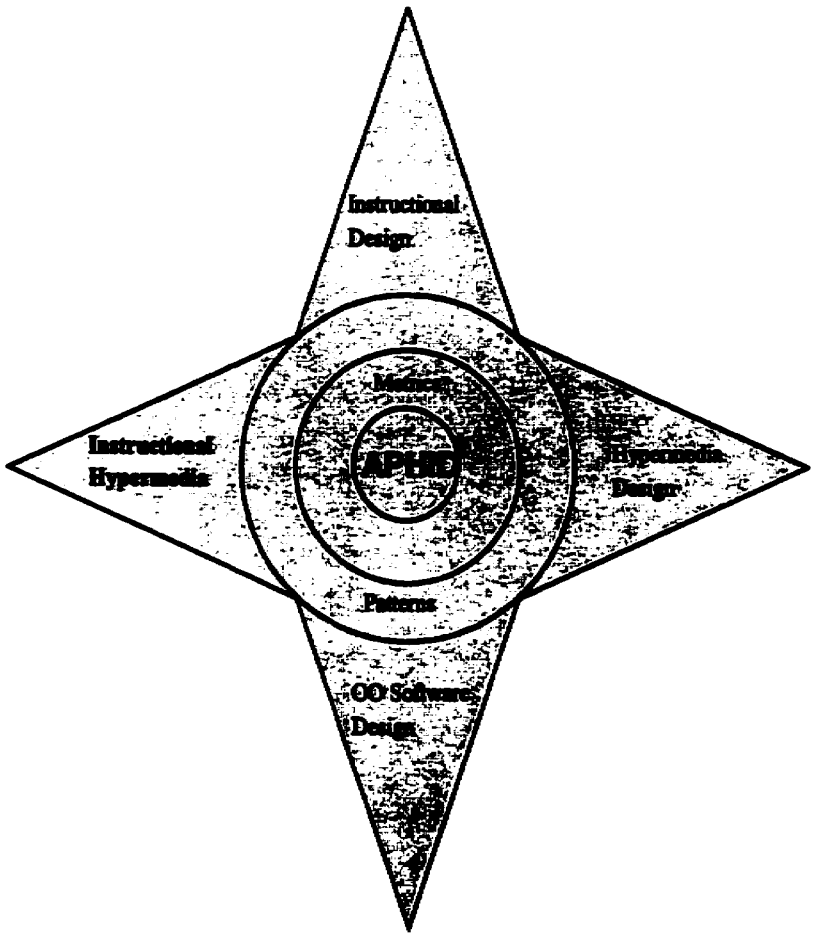


Figure 1-1 Scope of Research Project

develop the APHID software and a more detailed description of the approach. Chapter four presents a detailed description of the design of the prototype software. Chapter five describes a study done to evaluate the effectiveness of the APHID approach and presents the results of that study. The final chapter summarizes the research and proposes some directions for future work in the area.

Chapter Two

Models for Analysis and Design

Models are used by professionals from many disciplines to illustrate the details of their plans. Models are useful because they enable the designer to focus on salient details without cluttering the analysis with irrelevant facts. Models are controllable, meaning that the uncertainty of the real world does not affect the designer's model. All uncertainty can be assumed away, at least for early phases of the analysis and design process. Models have the added advantage of improving communication between persons involved in the design.

This chapter explores the use of models for three different domains utilizing analysis and design: software, hypermedia and instruction. The chapter looks at the analysis and design process within each of these domains and draws some parallels between them. First, the area of object-oriented (OO) analysis and design is explored, noting the types of models that are used within that process. Second, analysis and design of hypermedia is examined and the construction of hypermedia applications is compared to the process used for OO development of software. Third, the process of creating instructional materials is investigated. Although the field of instructional design seems, at first, much different from that of either software or hypermedia design, the processes are similar and the models used in the latter are compatible with those required for the former. The chapter concludes with a discussion of optimal analysis and design conditions for instructional hypermedia materials.

2.1 Object-Oriented Analysis and Design

Software development takes many forms and follows many processes. Each of these different processes is suitable for differing sorts of software. Object-oriented analysis and design is emerging as one of the most flexible methods of designing robust, modular

software. The process involves the use of several models, as well as the application of patterns to solve well-understood problems.

Part of the appeal of object-oriented development is that it provides an easy method of encapsulating parts of the system under consideration. The classes in a well-designed object-oriented system have clearly defined functionality and purpose within the entire system and are part of the structure of the system, yet they remain entities unto themselves (Meyer 1997). The process involved in creating an object-oriented software system transfers well to other types of analysis and design as well. Designers have the same goals regardless of the domain, to try to design components that work together without compromising autonomy or modularity. The realization of these goals begins with modelling the system under consideration to better understand the problem and the possible solutions.

2.1.1 Models for OO Software Development

Proponents of Object-oriented Analysis and Design (OOAD) have created many different types of models for software development. In recent years the differing methods have merged and now two distinct methods remain: the Unified Modelling Language (UML) (Fowler, et al. 1997; Rational Software Corporation 1998), and OPEN (Henderson-Sellers 1998). Both of these methods make use of models to guide the development process. The UML is a data-driven modelling language, suitable for applications where the data determines the behaviour of the application. OPEN uses a responsibility-driven modelling language (OML), more suitable for applications where the intended function of the application is clearly defined and the data is secondary (Firesmith, et al. 1998). The models created in both languages are similar and are one of three types: static, dynamic or presentation.

A static model defines the structure of a software system without defining how that system behaves. The static structure consists of the data types and their structure and how they relate to one another. The static model does not include any description of the behaviour of the different kinds of data or the services required to perform the work of

the system. The static model consists, in part, of a class diagram that details the classes (or concepts) involved with the software.

A domain model is a representation of the concepts and ideas that comprise the problem domain. Traditionally, the domain model was the province of database designers and was realized in the form of an ER diagram (Chen 1976). A domain model consists of concepts, attributes and associations. Concepts represent the ideas within the domain. Each concept may have attributes that give more crucial information about the concept, and associations represent the relationships that exist between concepts.

Within the UML, the domain model is called a *Conceptual Model* and is constructed to contain all the concepts within the domain that are relevant to the software being developed, including those that may ultimately not be part of the software system. A domain model is crucial to understanding what sorts of interactions may be necessary within the software system. It is also valuable as a means of defining the requirements for the system and communicating those requirements to users. As the design evolves, the static model expands to become a description of the classes that make up the software system including their attributes, operations and associations. It details the relationships between the classes, including inheritance relationships. The static model may be organized by packages and may make use of templates, interfaces and stereotypes to clarify the meaning. Usually the static model shows relationships between components, but not the interactions required to perform specific functions within the software system. Interactions and descriptions of system functions are usually described in a dynamic model.

The dynamic model of a software system illustrates the behaviour of an application. It represents the interactions that occur between the objects in a software application. Because of software complexity, all interaction cannot be shown on a single model, so the dynamic model is frequently organized around the intended uses of the system, or around the functions that the system will offer. Within UML the dynamic model for Object-oriented systems includes collaboration diagrams, which illustrate how objects collaborate to accomplish the tasks associated with the software, activity diagrams,

which simplify the illustration of choice points within a process, and state diagrams, which show the possible states of particular objects (Odell 1998; Fowler, et al. 1997).

The presentation model describes the look and functionality of the user interface. Most object-oriented notations do not explicitly define a presentation model, rather the design of the user interface is a separate process. It is suggested that the designer should make use of rapid prototyping, screen design tools and storyboards to create a presentation model. Regardless of the method for creating the presentation model, the finished model demonstrates what the application will look like and how users will interact with it. If a prototype is created, potential users may see what functionality the application will have without the need for the functionality to be fully implemented.

2.1.2 Patterns for Software Development

Patterns are used within the software community to codify analysis and design solutions to common problems. These patterns are examples of good design principles that can be applied in a particular context (Gamma, et al. 1994). Sets of patterns that are related and recur as solutions to particular problems form pattern languages that can be used to describe the problem solution. A design pattern describes a solution that has been found through experience, not one that is new or theoretical. Design patterns are intended to communicate the understandings developed by experienced software developers to novice and less-experienced people.

Design patterns are concise, clear descriptions of solutions to common problems that are known to be effective. A design pattern commonly consists of a textual description of the context, the problem and the suggested solution. Sometimes a design pattern is expressed more formally as an algorithm or fragment of code. It should present a description of the forces to be considered before adopting the patterns as well as the costs associated with the pattern. For instance, one well known pattern, the façade, suggests motivation and guidelines for providing a uniform interface to a subsystem of a software application (Gamma, et al. 1994). The façade pattern is used frequently for providing interfaces to storage classes such as database drivers. Common interfaces for software subsystems are not a new idea, but represent good programming practice that

has been around for several years. The pattern just records the motivation and procedure for taking advantage of that knowledge.

The use of design patterns in a software project facilitates communication between project members as well as serving as scaffolding for communication between new hires and more experienced employees. They assist designers to create robust abstract designs and the designs have built-in points for expansion. In other words, the design pattern takes into consideration possible ways that clients may wish to expand their software, and suggests a design that makes it relatively easy to accommodate relatively major changes to the software. At the same time, the flexibility built in to a system designed using patterns is constrained to specific classes and portions of the code, which restricts maintenance programmers to making modifications to a few classes. Although this may seem like a disadvantage at first, the result is code that is more understandable and less fragmented (Cline 1996).

2.1.3 Software Validation

The term validation, when applied to software development generally means a series of tests designed to show that the software built conforms to the contract or specification that was agreed to at the beginning of the software project. Sometimes validation also involves alpha and beta testing with users to determine how the users react to the software (Pressman 1997).

More recent validation strategies are centered on studies with users of the software. Such studies are less focused on whether the software does what it was intended to do, and more focused on whether users can use the software successfully. This form of validation is called usability testing and can either be done automatically (by using evaluation software on the software to be tested), empirically (with real users), formally (with usability metrics), or informally (based on heuristics and rules of thumb) (Nielsen 1993). When evaluating usability, automatic methods don't work well, and formal methods are difficult to apply. Most software usability is evaluated using a combination of empirical testing and usability inspection methods (Nielsen & Mack 1994).

Empirical evaluations can be effective, but experiments must be carefully designed to measure the desired aspects of the software and controlled to avoid contaminating results. Additionally, it is often difficult to find a suitable set of homogeneous users with whom to conduct experiments (McGrath 1995). Usability inspections offer a cost-effective method of evaluating software usability. One inspection method, heuristic evaluation, can be performed with a few evaluators and a short list of usability criteria (Nielsen 1993)

2.2 Hypermedia Analysis and Design

The process of developing hypermedia applications is similar to the development of any software application. Like software designers, hypermedia designers desire to produce well-designed, easy to use applications that meet the needs of users while minimizing cost and production time. A successful development process for hypermedia should include conceptual (or data) modelling, navigational design modelling, a model for run-time behaviour, user-interface design, and a method for moving from design to implementation (Christoldoulou, et al. 1998). Many different processes and notations exist for the development of hypermedia applications. Not all of these different approaches are based on an object-oriented metaphor, but most still use the same basic steps of analysis, design, and implementation.

2.2.1 Models for Hypermedia Design

Most analysis and design processes require some sort of model. Most hypermedia designs use much the same models as typical OO methods: data model, dynamic model and presentation model. The design process for well-designed hypermedia typically involves first selecting a topic, then creating a data model of the domain to better understand the structure of the information. Often the last step in the design is the creation of a presentation model that describes the components the user will see. Applications developed from such a design provide navigation for the user based on the structure of the data, using the elements defined in the user interface model.

This sort of design process works well in situations where the information supplied by the application is "reference" in nature. That is, the information has the same structure for a number of different instances, like a reference book on some topic. Web sites for museums are often structured in this fashion (Royal Tyrrell Museum 1997). The resulting application gives the user navigational ability based on the structure of the data, which is appropriate for many different navigational purposes. For many types of users, this may be the only sort of navigation required.

2.2.1.1 Domain Model

Hypermedia design processes that include a formal model of the data often work best in domains that can be described with a schema-like representation. These sorts of domains are composed of discrete 'objects' that have relationships with other objects and the relationships can be documented and described. The number of object types is limited, but there are usually many instances of each type of object. One common example is a personnel information system that provides information about an organization, its employees and departments. (3 classes, many instances).

Two examples of hypermedia design methods with this characteristic are the Relationship Management Model (RMM) (Isakowitz, et al. 1995) and the Object-oriented Hypermedia Design Method (OOHDM) (Schwabe, et al. 1996). RMM was specifically designed for this type of domain while OOHDM utilizes the domain modelling techniques of OMT (Rumbaugh, et al. 1991). Both RMM and OOHDM base the domain model on aggregation and abstraction hierarchies, which are most applicable to the structured type of domain mentioned previously. Such a model is less useful in a situation where each type of data (class) is represented once, but the number of different types is large.

2.2.1.2 Navigation Model

The dynamic behaviour of a hypermedia application is not exactly like that of an object-oriented application. A dynamic model of an object-oriented application shows collaborations between classes to accomplish a task. In the case of hypermedia applications, the dynamic model should represent the behaviour of the application in reaction to input by the user (selecting a link or the button that invokes a javascript

segment, for instance). This behaviour is partially represented in what is known as the navigation model. Diagrams similar to UML collaboration diagrams have been used to represent this behaviour. Most of the behaviour can be inferred from the data model and the relationships on the data model.

RMM uses the relationships in the data model to suggest placement of hyperlinks within the hypermedia application. The associative relationships in the ER diagram are represented as navigational structures in the hypermedia application. Three possible navigational structures are identified: conditional indices, conditional guided tours and conditional indexed guided tours (Isakowitz, et al. 1995).

Within OOHDM, navigational design is built using the conceptual model (domain) as a guide, but the method allows for more than one navigational model for a single domain model (different views on the database). A set of navigational classes are defined, similar to the navigational elements defined for RMM. Navigational classes include nodes, links, and access structures such as guided tours and indices. Node classes contain a sub-set of the information about a concept in the conceptual model as well as anchors that indicate associations with related concepts.

A navigational context is simply a set of navigational classes and (possibly) other embedded navigational contexts. A navigational context is used to provide navigation through the information space based on one (or more) characteristics of the nodes. For example, a personnel information system may wish to allow users to browse the list of employees based on which department they work for, or based on the project to which they are currently assigned. Each of these options is a different navigational context. A navigational context class defines how the node class appears within a certain context. The notion of navigational contexts has no direct equivalent in OO analysis and design.

OOHDM specifies the dynamic behaviour of the application using modified state transition diagrams called Navigation Charts. Navigation Charts describe the rules for transformations that occur during link navigation (for example, does a node appear in the entire window, or does it pop up beside the existing window) (Schwabe, et al. 1996).

2.2.1.3 Presentation Model

RMM develops the idea of a 'slice' as the preliminary presentation model. A slice represents some subset of the information associated with a class, presented to the user in meaningful chunks. The various slices of the class are grouped together with hyperlinks to present all the required information about the entity. RMM also suggests some guidelines for creating the interface based on the data model. For instance, in the case where an entity has a 1-n relationship with another entity, one recommendation is that the hypermedia application provide a guided tour between the entities. However, if the number of instances on the multiple side of the relationship is large, an index is recommended instead. The slice diagram is combined with a more traditional storyboard or layout diagram for the user interface to complete the presentation model (Isakowitz, et al. 1995).

Within OOHDM, the presentation model uses an Abstract Data View (ADV). ADVs are facades that specify the user interface component for a particular piece of information without giving any specifications about the data itself. The data is managed by an Abstract Data Object, which is in turn managed by the ADV. An ADV is an object, in the OO sense, that can respond to user events and has attributes that define its position, color, font, etc. A configuration diagram is created (within the abstract data model) to indicate the collaborations between the different ADVs that make up a user interface (Rossi, et al. 1995). The specification of Abstract Data Views does not completely define how the user interface will look (in terms of layout) and must be combined with a storyboard-like diagram to completely illustrate the layout.

2.2.2 Patterns for Hypermedia Applications

Design patterns have the same function when designing hypermedia applications as they do for software design. Patterns for hypermedia describe known solutions to common problems in the design of hypermedia. Generally, the description of hypermedia design patterns follows the same format as for software patterns. Patterns for hypermedia applications must, however, take into consideration many factors that do not affect software design such as the variable user paths through the application and the

ambiguity often associated with describing the data. Design patterns for hypermedia applications can be classified as either navigational, interface or structural (Garrido, et al. 1997; Rossi, et al. 1997; Bernstein 1998).

Navigational patterns describe methods of providing navigational choices to hypermedia users while maintaining the integrity of the models describing the hypermedia application. Navigational patterns generally describe single 'units' of information be that a node, a page or a topic. One navigational pattern, navigational *context*, describes how the context of a particular information item along with contextual hyperlinks can be provided to the user (Garrido, et al. 1997). This is accomplished using the *decorator* pattern (Gamma, et al. 1994) without requiring the information node to maintain knowledge of its context.

Interface patterns are mainly concerned with the effective combination of elements to create a single 'page' of information for the user. The patterns address such things as the number of elements to present, the layout of elements and how to best communicate to the user what the various widgets on the page are intended to do. An example of an interface pattern is *link destination announcement* which suggests that links should provide information about their destination to avoid the problem of users triggering links to undesired destinations (Nanard & Nanard 1998). Like navigational patterns, interface patterns offer suggestions for structuring and organizing individual units of information.

Structural patterns take a larger view of hypermedia applications and offer suggestions about how the intended 'path' through the application might best be designed. Bernstein (1998) identifies several structural patterns. Each of the identified patterns has advantages and disadvantages depending on the author's intended use of the hypermedia application. For instance, the *mirrorworld* pattern describes a hypermedia application that is written from two (or more) distinct points of view. Each point of view has the same structure and (hopefully) cross links allowing the reader to examine the two points of view more or less in parallel. Other patterns describe different sorts of paths through the hypermedia application.

Similar to software design patterns, patterns for hypermedia applications are used to guide the creation of the models that describe a hypermedia application. But, unlike most software patterns, hypermedia application patterns can also be used by algorithms that automatically create hypermedia applications from databases. The notion of reusable templates is one way of implementing the automatic generation of hypermedia using design patterns as a guide (Nanard & Nanard 1998). For their project, source data was described using SGML. The SGML files were processed using an SGML parser, which created a derivation tree for the hypermedia. The tree, applicable design patterns and navigation and presentation specifications were then submitted to a page generator that produced the final hypermedia page instances.

Patterns have been identified that codify the use of several different 'components' for hypermedia applications (Schwabe, et al. 1996). These patterns are concerned with the analysis and design of a hypermedia application from a static perspective. Many of the models associated with OOHDM identify and document the structure of a hypermedia application without addressing the behaviour of the application. The patterns associated with such models also identify the structure, but not the behaviour. In many cases, some of the behaviour can be inferred from the structure, but that is insufficient for many purposes.

Presently most efforts to describe design patterns for hypermedia applications have focused on navigational and interface patterns. The specification of structural patterns is more difficult because the perceived structure of a hypermedia application is dependent upon the reader's approach to the document. Structural patterns are most useful when the author of the hypermedia application has a preconceived notion of how the application 'should' be used. Such preconception is common when the author is writing narrative hypertext (fiction, non-fiction and poetry) or when the hypermedia application is instructional in nature.

2.2.3 Metrics for Hypermedia

Hypermedia is not validated in the same way that software is validated, but similar goals drive the search for metrics related to hypermedia. Traditional software validation is

concerned with determining if the software product is right for the intended task. Typically this involves testing of the various functions of the software and of the user interface to ensure that the functions can be used correctly.

Hypermedia metrics are concerned as well with these two dimensions, but hypermedia metrics must also involve an evaluation of how the use of the hypermedia application affects the user's work. Specifically one must evaluate if the hypermedia application will be used, if it is adaptable to the different tasks that users will have, if it is cost effective to construct, if the intended users have the skills necessary to use the application and if the application is 'good enough' (Wright 1991).

In addition to common software engineering approaches to validation, creators of hypermedia applications use several methods to evaluate their work. The approaches are varied, and cannot be easily classified. They generally fall into one of three categories: experiments with users, semantic analysis, and structural analysis. Although each approach yields different information about the hypermedia application, the overall goal is still to determine if the hypermedia application is good enough, and subsequently to use the information to make the application better.

The next few sections discuss methods that are normally used to create hypermedia applications that conform to particular characteristics. The same methods of generating 'good' hypermedia applications can be used to evaluate existing hypermedia as long as the data required to perform the required calculations is available.

2.2.3.1 User Studies

Any effective software validation exercise must include some sort of evaluation of the use of the software by users. As opposed to software validation, the evaluation of hypermedia applications requires even more attention to how the user works with the application, because rarely is the hypermedia application specified in enough detail to validate from specifications. As a result, significant effort is expended providing tools and techniques for hypermedia users to try, and then analysing their effects on the user's effectiveness with the hypermedia.

Earlier research into how users interact with hypertext focused on the process of choosing and traversing hyperlinks to determine if the hypertext 'worked' for the users (Wright 1991). In one study, novice users of hypermedia identified eight different types of links within hypermedia (e.g. sequential, comparative, causal), however little has been done to determine which of those types of links makes the hypermedia application work more effectively (Harmon & Dinsmore 1992). Much of the modern work on user interactions with hypermedia comes not from researchers wishing to validate hypermedia applications, but from researchers of user modelling who wish to understand user interactions with hypermedia applications in order to create a model of the user. Although at first this does not seem to have much to do with the measurement of hypermedia applications, the whole purpose of the user model is to allow customization of the application to the user. In other words, a user model is intended to be used to make the hypermedia application better (for a particular user) so the methods used to learn about user modelling in hypermedia can also be used to evaluate hypermedia applications.

Validation of user interactions with hypermedia is limited by what can be observed, either directly or through some sort of monitoring. One observable characteristic of users is the path they take through a hypermedia application. User browsing patterns form the basis of many efforts to determine the effectiveness of a hypermedia application. A record of user browsing behaviour through a hypermedia application allows comparison of different users and classification of those users into groups. One method of classifying users relies on a *longest common subsequence* algorithm for determining the similarity of one path through the hypermedia to another path (Sun & Ching 1995). Different users of the same hypermedia application should have similar paths through the hypermedia, if their purposes and skill levels are relatively equal. If their paths vary markedly, it is possible that the hypermedia application is not quite right.

A second approach to using browsing history as a tool for improving a hypermedia application considers not the path through the hypermedia, but the material viewed by the user (Yan, et al. 1996). Each page the user accesses, along with the approximate

time spent on the page is represented as part of a vector. The vectors from different users are clustered and the clusters are used as classifications for the users. From a user-modelling viewpoint, the user of the hypermedia system is monitored and classified and then the hypermedia application can be modified to better suit that user. From a hypermedia validation viewpoint, users with the same purpose and background skill set should fall into the same category when the clustering is done.

2.2.3.2 Semantic Analysis

In addition to determining that the hypermedia application works as intended for users, it is important to ensure that the application conveys the intended meaning cohesively to the user. This means that each node in the hypermedia application must have links to other semantically related nodes, but usually not to nodes that are semantically unrelated (or to few unrelated nodes). Most often the creator implicitly defines the meaning of a hypertext application as links are created in each individual node. The semantic analysis of manually created hypermedia systems is usually done by traversing each link to determine if the link fits with the intended meaning of the application.

The idea behind semantic analysis is to determine if two nodes of the hypermedia system are similar enough to warrant a hyperlink between them. Semantic analysis of hypermedia applications has all the same challenges as full text information retrieval: the indexing of the document set is time and space inefficient, it is difficult to measure effectiveness, and formulating queries is often hard for users (Salton, et al. 1994). Often, semantic analysis is done when the hypermedia is being generated dynamically, perhaps as a result of a user query (Bodner, et al. 1997). The analysis is done using full text retrieval engines, such as INQUERY (Callan, et al. 1992) or more common vector based engines. In this case the analysis is done to determine the similarity of a document to a query provided by the user. The similarity measure determines whether or not the document is presented as a result to the query.

Another use for measures of similarity is to determine the similarity between two documents. (Broder, et al. 1997) use a clustering algorithm on the syntax of documents along with measures of syntactic resemblance to identify documents that are syntactically similar. This information is used to locate documents from the web that

are duplicated in many places. While this approach does not lead to an identification of semantic closeness, the clustering approach to grouping large quantities of documents is interesting (the test case was 30 million web pages) and it is likely that some measure of semantic closeness could be used in a similar fashion.

Any automated approach to measuring and evaluating the semantics of a hypermedia application is going to rely on measures of similarity. Presently semantic indexing is not being used to validate hypermedia applications, so the issue of which approach is most suitable is an open research question. The full-text retrieval approach is time consuming, but may be simplest for a validation activity that does not require real-time responses. On the other hand, a clustering approach may be more effective.

2.2.3.3 Structural Analysis

By far the simplest metrics to calculate for hypermedia applications are those that use the application structure. A hypermedia application can easily be represented as a directed graph and all of the algorithms for working with directed graphs can be applied. The metrics found through such analysis can be used to validate the hypermedia application.

The most obvious characteristic that can be determined from the topology of the directed graph is the number of arcs connected to each document in the application. Many analyses of hypermedia topology begin with an examination of in-degree and out-degree because they are measures that are straightforward to determine. Unfortunately, a simple count of degree does not give much information about the documents in a hypermedia application. Intuitively, one would expect that documents with large in-degree would be documents with high relevance to the application. This approach has difficulties however, not the least of which is the problem of determining what a 'large' in-degree might be. Large is a relative measure that changes with the application. Nonetheless, topological analysis continues to be the focus of many researchers, possibly because the algorithms for analyzing graph structures are well known and well understood. The remainder of this section looks briefly at several different approaches to measuring hypermedia through structural analysis.

A topology matrix is an adjacency matrix that represents a collection of web pages. A topology matrix can be used in conjunction with usage information about frequency and entry point to create a feature vector for each document in the collection. Feature vectors include information about size, in-links, out-links, frequency of requests, entry points, textual similarity and depth. Categories of documents, such as home page, reference page, and index page, are identified prior to the analysis and the expected weights for documents in that category are set. When the process finishes, the documents in a collection are classified according to the previously identified categories (Pirolli, et al. 1996).

Mukherjea and Foley calculate the importance of a web page using connectedness measures (Mukherjea, et al. 1995). In addition to in-degree (I) and out-degree (O), two additional notions of connectedness are introduced to determine the importance. *Second-order connectedness* (SOC) is the number of pages that can be reached from a page by following at most two links, and *back second-order connectedness* (BSOC) is the number of pages that can reach a particular page using at most two links. The importance of a page p is given by: $imp(p) = (I_p+O_p) * wt1 + (SOC + BSOC)*wt2$, where wt1 and wt2 are arbitrary weights chosen so that (wt1 + wt2 = 1).

Restricting topological analysis to explicit hyperlinks may miss some important structural information contained in the hypermedia collection, especially within the context of the web (Spertus 1997). A set of heuristics can be used to mine extra structural information from within a web site to determine how documents relate structurally to one another independent of the hyperlinks they contain. Several classes of links are used to make inferences about the type of page that is referenced. For instance, spatial links are used to guess at the similarity of two documents. One heuristic states: "If URLs U1 and U2 appear 'near' each other on a page, they are likely to have similar topics or some other shared characteristic" (Spertus 1997).

This heuristic could be used, for instance, to determine that the URLs in a bulleted list on a web page were all related to one another in some way, but that they were not necessarily related to the URLs in other bulleted lists on the same page. A number of different heuristics are used to identify a variety of implicit link types in hypermedia

pages. The different kinds of links are identified using SQUEAL, a relational database system for managing WWW structural information (Spertus 1998). The process can be used to determine similarity among documents or to find documents with specialized information within a hypermedia application.

A potentially useful attribute of hypermedia applications is the measure of how far a page is from any other page in the application. One can calculate a distance matrix for any directed graph by a number of shortest path algorithms (Horowitz & Sahni 1978). The difficulty with using standard shortest path algorithms is that the entries for documents that are not connected are infinite. Infinite numbers are problematic in any sort of calculations with the distance matrix.

One solution to the infinite distance problem is to create a converted distance matrix that substitutes the infinite entries for a carefully selected constant (Botafogo, et al. 1992). From the distance matrix one can determine the *centrality* of a node. A node with high centrality must be able to reach all (or most) of the other nodes in the hypermedia and must be relatively close to the other nodes. Two types of centrality can be calculated from the converted distance matrix, *relative out centrality (ROC)* and *relative in centrality (RIC)*. Relative out centrality is a normalized summation of the distances from a page to all other pages. Similarly, relative in centrality is the normalized summation of the distances from all other pages to a particular page (Rivlin, et al. 1994).

The centrality of nodes can also be used to help automatically determine the hierarchy structure of the entire hypermedia application (Botafogo, et al. 1992). First, a highly central node is selected as the root node. Next, hyperlinks within the application are classified as either *structural* (part of the hierarchy) or *cross-referential* (not part of the hierarchy). The classification is accomplished by finding the spanning tree of the graph representing the application using a bread-first search. All arcs that are part of the spanning tree (the shortest paths) are hierarchical links, other arcs are cross-referential links (Botafogo, et al. 1992).

Several alternates to hierarchical structure exist in hypermedia. One possibility is to view the site as a collection of *hub* pages that reference a collection of *authority* pages.

A good hub page is one that points to several authority pages, and a good authority is one that is pointed to by several hub pages. Authorities and hubs are found iteratively using in-degree and out-degree to adjust the authority and hub weight of each page. The iterative weight-assignment algorithm was used on the results of a broad Alta Vista search to find pages within the results that were more authoritative than other pages. The results from this test were good; the algorithm correctly identified authority pages most of the time (Kleinberg 1998).

2.3 Instructional Design

Instructional Design is the process of selecting and organizing (and evaluating) methods of instruction and materials for learning (Reigeluth 1983; Seels & Richey 1994). Good instructional design results in a set of materials that helps a learner efficiently and effectively reach the learning goal. Instruction can be separated into two processes: understanding the content and understanding the presentation. (Wasson (Brecht) 1990) During instruction, educators use their understanding of the content and presentation to create a learning environment for students. Instruction is always a planned activity (Corey 1971). Even impromptu lessons must be prepared for and planned for, although much of that planning comes as a result of the basic preparedness of a good instructor.

Learning theorists are divided about the 'best' way to provide instruction. Objectivists claim that knowledge can be broken down into concepts, facts and behavioural steps. Instruction, to an objectivist, is simply the organized presentation of the various knowledge components. The presentation must consist of some opportunity for the learner to practice the skills with guidance from the instructor. Constructivist learning theorists, on the other hand, believe that understanding of subject matter is constructed individually by each learner and that of paramount importance is the context in which the learner works, the authenticity of the learning task, and collaboration with other learners (Jonassen 1991; Wasson 1996). And finally, cognitivist learning theorists are more concerned with the meaning of learning and with the construction of a mental model or schema about the domain of instruction (Chitrenky 1996). For example, an objectivist might be comfortable creating drill and practice software for children to learn

mathematics because the presentation of the content could be strictly controlled and the practice could be monitored and guided carefully. A constructivist might think that mathematics drill and practice lacks authenticity and that learners would be better to spend their time in tasks that are more consistent with real-world applications of mathematics. A cognitivist would want to help the learner understand the components or ideas that make up the particular mathematical topic and to ensure that the learner built up a notion of how that topic is related to other mathematical topics.

Most computer-based tutoring systems are designed to teach procedures and facts and based on an objectivist view of instruction. Many 'exploration' type educational computer programs are designed to teach concepts and ideas, and to assist learners to build up their knowledge of the domain through free exploration of the computer-created environment (Oliver 1995). Many of these free exploration type of environments are based on constructivist and cognitivist views of learning.

2.3.1 Models for Instructional Design

Instructional systems development is the design of development of instructional materials. It consists of more or less the same basic phases as computer systems development: analysis, design, development, implementation, and evaluation (Merrill 1996). Instructional Design theories abound, each with its specialized approach to instruction. Theories on instructional design attempt to describe how to approach instruction in a methodical and organized fashion. Traditional approaches to instructional design parallel traditional waterfall approaches to software design, often with the same results, a working system that doesn't meet the needs of the intended audience (Wilson, et al. 1993). As a result of both the limitations of a linear development approach and the divergence of instructional design theories, many new approaches to creating instructional materials have been proposed. Some of the new approaches are encapsulated as theories on instructional design which provide guidelines about how instruction should be done, and some are given as methods or technologies to assist with actually creating instructional materials (Wilson 1997). Regardless of theory, the design of instruction requires, as a minimum, that the designer

consider the content, the sequencing of the material, and the presentation. Even these minimalist considerations result in models that parallel the models used for OO software development (conceptual, dynamic and presentation).

2.3.1.1 Content

The content model for an instruction corresponds in part to the conceptual model for a software system. Many possibilities exist for creating content models including the common technique of building a concept map for the domain of instruction.

A concept map is a representation of the concepts in a particular domain. These concepts, as well as their relationships to other concepts can be depicted in some meaningful (usually graphical) format (Kremer 1997). As a minimum, a concept map contains concepts and connections. An example of a minimalist concept map is shown in Figure 2-1. More often, the relationships include, but are not limited to, aggregation and abstraction relationships. This makes possible the expression of specialized relationships between concepts. Another difference from conceptual models of software systems is that the notion of concept is not as rigorously defined as the notion of class. A concept can be similar to a class, but a concept could also encompass instances from a number of classes. For instance, the concept of *data-structures* contains instances like *stack*, *tree*, *list* etc. It would be relatively simple to create a data structure class and represent each of the instances given as members of that class. The concept of *sorting* however, might also contain the instance *tree*, but would also contain *bubble sort*. Tree and bubble sort are clearly not the same class, but do belong to the same concept within a certain context. Allowing for many different levels of abstraction facilitates a finer description of the semantics of the domain. A concept map allows discussion about some of the semantics associated with a particular domain without (necessarily) needing the data elements that instantiate the concepts.

A concept map, for an instructor, represents a problem domain that will be partially presented to learners during the course of instruction. Different models of instructional design propose different steps and activities to present the map to the learner. For example, in one model, the key activities during instructional design are to determine how many of the domain concepts to present to the learners, in what order to present them, to what level of detail they should be presented, and what activities should accompany the presentation (Merrill 1998). The purpose for the instruction, along with information about the learner's needs and abilities determine the quantity, sequence,

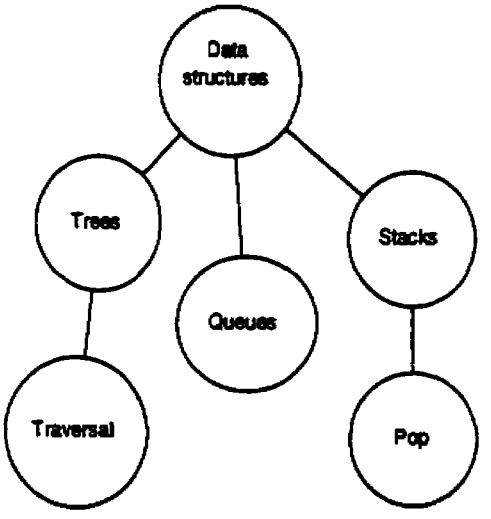


Figure 2-1 Example Concept Map

depth and activities. An instructor who desires to provide a review will teach differently than one presenting new material, even to the same students. However, an instructor who is providing a review will have much the same approach to communicating the concepts within the instructional domain as any other instructor teaching a review.

An alternate method of modelling content for instructional purposes is demonstrated in Instructional Transaction Theory (ITT). The models used in ITT are intended to add richer semantic information to the domain model for an instructional hypermedia system (Merrill 1996). When designing an instructional system using ITT, the designer creates knowledge objects that are similar to concepts in a concept model. The major difference is that knowledge objects contain information about other objects that are related, in a

variety of ways, to the current object. It also contains descriptions of activities and processes that are associated with the knowledge object (Merrill 1998).

2.3.1.2 Instructional Strategies

Instructional strategies are the implementation guidelines for instructional design theories. Each instructional designer has a favoured design theory along with a preferred set of instructional strategies. After the content is selected, the essential elements of instruction are guidance, practice and presentation (Merrill 1998). The complexity of instructional strategies is as varied as the number of strategies. Some strategies are more conducive to construction of meaning, while others favour direct instruction (Wilson 1997). The differences between the strategies center around what kind of material is being presented, when to provide guidance to the learner, what form the guidance should take, how much and what kind of practice to give, and how best to present new knowledge to learners.

The nature of the content is possibly the most important aspect to consider when choosing an instructional strategy. If the material is new to the learner, or is of a sort that involves practicing small steps or procedures to accomplish a larger goal, an instructional strategy that is instructor-directed is indicated. However, if the learning goal is to develop high-level concepts or understandings, an instructional strategy focused on knowledge construction is recommended (Oliver 1995).

The amount of learner guidance given can range from rigidly controlled sequencing of information to occasional suggestions about which information might be interesting or useful to pursue next. The amount of guidance required depends heavily on the content. A learner trying to master a complicated sequence would require more guidance than one who was simply browsing for interesting information.

Presentation of instruction is partially determined by the instructional strategy chosen and partially influenced by basic usability considerations. An instructional strategy that suggests that material be given to the learner in small sections in a prescribed sequence will also suggest a presentation style that is compatible with small information elements. Critical to learner comprehension are the structure of the material, and the readability of

any text. That is, the material should have a readily identifiable structure that learners can use to form concepts, and any text provided to learners must be easy enough for them to read, and ideally provide aids to comprehension, for example, a glossary (Oliver 1995).

2.3.2 Patterns for Instructional Design

The education community has begun to describe educational practices using patterns (Anthony 1995; Manns 1998). Patterns are suited to the description of educational practices because, like software development, there are few exact rules for creating good instruction but many good practices that evolve out of the experiences of educators. The good practices are handed down verbally to novice instructors, but a pattern-based description lends some formality to the sharing of information. Educational patterns describe approaches to structuring instruction within a classroom setting. They provide suggestions for instructional strategies that work for particular types of instructional goals given content with certain characteristics. For instance, one pattern describes how to approach the teaching of material that consists of a number of equally important concepts, all of which are interdependent (Anthony 1995). The instructional goal for this pattern is an in depth understanding of the material and the driving characteristic of the data is that it is interdependent.

One common instructional sequence contains nine identifiable elements or steps (Gagne, et al. 1988). Educators using this sequence first motivate the learning then provide an overview of what is to be learned. Learners are then given a review of the knowledge they already have prior to the actual instruction of the new concepts. After instruction the learners participate in guided practice with feedback from the instructor, are tested on their new knowledge and then either remediated or given enrichment. While this sequence is used extensively by educators, it is less commonly used by designers of hypermedia instructional materials (Ritchie & Hoffman 1996). These kinds of sequences can be represented as patterns and used by hypermedia application designers to create instructional hypermedia and to communicate with instructional designers about what the hypermedia application is designed to accomplish.

2.4 Supporting the Design of Instructional Hypermedia Materials

An educational hypermedia application is any hypermedia application intended to provide new knowledge to the user. Instruction, however, is more a process of guiding a learner through a set of information by providing the information in pre-determined units in a specific sequence, punctuated with activities to support learning (Ritchie & Hoffman 1996). An instructional hypermedia application then, is one that provides a definite sequence of instruction with the intent of imparting a specific set of knowledge to the user. Instructional hypermedia applications are a subset of educational applications. Their design is similar to the design of any hypermedia application with the added requirements of providing an appropriate sequence through the hypermedia and ensuring that the units of information presented are appropriately sized for the intended learners and the learning goal.

2.4.1 Models for Instructional Hypermedia Applications

The identification of concept sequence and semantics is a process frequently forgotten or ignored in hypermedia analysis and design. It is, however, an activity that is required for the construction of instructional hypermedia applications. The identification of the knowledge to be taught and its characteristics is one of the two main steps in instructional design (Merrill 1998). While the knowledge is usually identified and modelled when creating hypermedia, the sequence of instruction for the individual pieces of knowledge often are not. In addition to conceptual, navigation and presentation models, designers of instructional hypermedia require a model that assists them with the design of the sequence. This model is called a *semantic model.*

A semantic model, for hypermedia designers, is essentially the same as a conceptual or domain model. For instructional design, however, it is slightly different. A semantic model is the model that allows the instructor to design the sequence of instruction to accomplish the instructional goal. One approach is to try to capture the activities associated with teaching in a separate model. This results in a broad model of instruction that includes information about teaching acts such as evaluation and practise, but often doesn't link the teaching elements directly to the content being taught

(Vassileva 1995). Semantic modelling is typically not considered when designing hypermedia applications. Usually the conceptual model provides enough semantic information. The navigational design can be adequately defined from the domain model and no further modelling occurs. However, if the navigation depends on the meaning of individual instances of concepts, rather than an abstract understanding of the domain, a more detailed analysis of the semantics must be performed.

This sequence can be represented as an addition to the navigation model for the hypermedia application or, the two types of navigation can be represented independently as *data model navigation* and *sequential navigation* (Frohlich & Nejdl 1998). Data model navigation is the navigation provided by the links within the conceptual model. Sequential navigation is navigation that is designed to present topics in a particular sequence. The sequencing of instruction is dependent upon the instructional strategy selected by the instructor. Given a particular instructional strategy, predictions can be made about appropriate sequencing of information in an instructional hypermedia application based on the learning goal selected by the learner or the organization inherent in the domain of instruction. Alternatively, the sequence can be imposed externally by the instructor.

In most traditional instructional situations, the instructor sets the sequence through a given set of material. An experienced instructor is skilled at modelling the instructional domain to identify the relevant concepts and examples and then determining the best sequence to present the information for a given student (or set of students). Instructors select the presentation sequence based on both the structure of the knowledge that comprises the subject area, the semantics associated with the individual pieces of knowledge, and the background of the student(s) being instructed.

The domain model for a hypermedia system considers the structure of the data in terms of associations between data elements of different types. It provides a general picture of the kind of data that makes up a domain, but does not provide an understanding of how the domain is best presented instructionally. In instructional hypermedia design, the semantics of the data, with respect to instructional sequencing, can be represented in a separate model, using a concept map (Paquette & Girard 1996). This semantic model

provides a bridge between the individual data elements, which are cumbersome to work with during the design process, and the other models of the application which provide an understanding of how the navigation through the application occurs, yet have no representation of the instructional requirements.

Presently few modelling techniques exist for creating a model of the semantics for instances of concepts. One case where a form of semantic modelling occurs is when documents are indexed to determine the similarity of two hypermedia documents. The indexing is usually done using some sort of measure of semantic closeness and the result is a representation of the semantics of an entire hypermedia application with respect to some base document (or other documents in the application). These types of models are most often used as information retrieval tools rather than as supports for application design. Most models of this type are really measures of syntactic similarity rather than semantic similarity due to limitations in full text recognition software (Broder, et al. 1997).

2.4.2 Guidelines for Instructional Hypermedia

Generally, guidelines for creating instructional hypermedia are the same as the guidelines for creating any hypermedia. The focus must be on good user-interface design and basic usability (Schneiderman 1997). The difference between instructional hypermedia and other hypermedia is that the former is intended to convey specific knowledge to the user of the hypermedia application. In other words, instructional hypermedia must educate or teach in addition to being a good hypermedia application.

Hypermedia applications are examples of constructivist and cognitivist approaches to learning combined. Learning of a topic is seen as the formation of a model of the subject by slowly building knowledge through working with the application (Eklund 1995). The application provides as realistic a set of information as possible to simulate real experiences with the information. The instructional designer must, from this perspective, carefully organize the instructional application so that it conveys to the learner the desired view of the domain of instruction without restricting the learner to a tour-like environment with no (or few) choice points.

The considerations for creating instructional hypermedia include an analysis of the content, selection of the instructional strategy, and consideration for the learner's background and capabilities. These are all issues that any instructor considers when creating instructional materials. Designers of instructional hypermedia must additionally consider issues of structure, navigation, orientation and fragmentation (Oliver 1995).

The structure of a hypermedia application is essentially one of three variants: linear, hierarchical or web. There are several variations and combinations of these three structures that are also used within applications. The instructional designer must choose the general structure that best supports the learning goal for the application. A linear structure is suitable for when the learning goal is specific, such as learning a particular procedure. The hierarchical and web structures are more suitable when the learning goal is more general. Variants on the three structures provide for goals that fall somewhere in the middle.

Navigation is a consideration that designers of printed instructional material deal with only tangentially. Instructional hypermedia designers must provide enough navigational opportunities in the application for users to use the application, yet must not clutter up the interface with so many choices that the user becomes confused or overloaded. For example, users need to have some method of maintaining their orientation within the hypermedia application. They need to know 'where' they are in the application relative to some known starting point or landmark. If the tool that helps them maintain orientation is complex or difficult to use, either they won't use it, or the overhead of learning to use the orientation facility will outweigh the benefits of having it in the first place.

The final consideration identified by Oliver (1995) is fragmentation. Fragmentation is essentially a readability issue, but is somewhat unique to hypermedia. Because information is presented as a series of nodes in hypermedia, often the context of association is lost when moving from one node to another. The result is rather like reading a disjoint set of information from small index cards. The cohesion of the narrative is lost when it is divided into nodes. Methods of coping with fragmentation

include using the information in the hyperlink to emphasize the association between the two nodes, or to display nodes together to maintain some context (Oliver 1995).

2.4.3 Creating Instructional Hypermedia

Design support for the creation of instructional hypermedia must provide assistance for designing and building hypermedia as well as for the pedagogical decisions that must be made. The systematic approaches to hypermedia design noted in Section 2.2 provide modelling support for data model navigation but not for sequential navigation. Conversely, many support tools for the design of educational hypermedia assist with sequential navigation design at the expense of developing a rich data model to allow for data model navigation (Frohlich, et al. 1997). A hybrid approach to the design and development of instructional hypermedia is needed.

As a minimum, the approach must allow the instructional designer to model the curriculum sequence as well as the structure of the data that will make up the application. The sequencing models must consider many different aspects of the learning task including the identification of the learning goal and the sequence of information presented to explain one sub-goal (Brusilovsky 1999). Since object-oriented analysis and design has a rich set of models capable of representing a wide variety of information, it seems plausible that a successful approach would draw upon the models from object-oriented analysis and design while incorporating current practices from instructional design.

The creation of instructional hypermedia requires that the designer be expert in both hypermedia development and instructional design. Software support for instructional designers falls into four categories: authoring systems for computer-based educational courseware, independent tools for specific design tasks (content modelling for example), expert systems to provide automation and advice for the design process, and general design tools that incorporate limited instructional design knowledge (Paquette & Girard 1996). Many systems designed to support the creation of instructional hypermedia, such as WebCT (Goldberg & Salari 1997), provide support for record keeping, learner

tracking and data management without providing any support for the design of the hypermedia application.

The most significant disadvantage of many instructional hypermedia design/authoring systems is the support of only a narrow set of educational strategies. Another disadvantage is that the entire development process for courseware is not integrated. MISA is a integrated toolkit for engineering learning systems intended to address the limitations of existing tools for the development of educational materials (Paquette, et al. 1998). MISA is much more than a development system for hypermedia applications, in fact it has no direct support for hypermedia development. It is more a computer-supported development methodology for instruction. It has strong support for the structured development of broad instructional courses, but less for the development of the components that make up a course.

MISA approaches the development of instruction from an object-oriented software engineering viewpoint. The development process is decomposed into a series of inter-related tasks that focus on the creation of a knowledge model, a pedagogical model that represents the learning events at a course granularity, and a media model that describes how the different types of media are created. The three models used in MISA correspond roughly to the conceptual, dynamic and presentation models used for OO software development.

MISA also contains no assistance for the designer about what makes 'good' instruction, just advice about how to document the creation of instruction. MISA cannot presently be used to guide an instructional designer through the creation of a good hypermedia application. As instructional development tools are created with an awareness of instructional design theory and offer more assistance to the designer, they also become less neutral in terms of how instruction is perceived. A tool that offers advice about sequencing of information and presentation styles has built-in instructional strategies that form the basis of the advice. As long as the instructional designer is comfortable with the strategies that are part of the design tool, it works well. Instructional designers must carefully select tools that match their own views on instruction.

The KBS Hyperbook system recommends four models for the development of adaptive instructional hypermedia applications: a domain model, a navigational model, a visualization model and a user model (Frohlich & Nejdl 1998). A static (non-adaptive) application can be constructed using all but the user model. A hyperbook is a collection of hypermedia documents, organized to allow both data model and sequential navigation. The data model navigation is determined from the conceptual model for the domain, which is created using an enhanced ER model. Multiples in a 1:m association have an additional key to indicate their relationship to the other multiples in the association. Navigation is determined from the domain model. 1:1 associations are translated into bi-directional hyperlinks. Hyperbook offers several different types of data model navigation through a 1:m association: indices, guided tours, indexed guided tours and cross-referenced indices.

Sequential navigation is provided in the form of a *reading sequence*, which is also determined from the domain model. A concept is chosen as the beginning of the sequence and a depth first traversal of the graph representing the domain is performed. The sequence of the traversal is the reading sequence for concepts. The sequence of objects in a 1:m (or n:m) association is determined from the additional key mentioned previously. The result is a hypermedia application that offers both navigation based on the data model for the domain, and a limited form of sequential navigation in the form of a reading sequence. The reading sequence represents only one specific instructional strategy. The hyperbook system does not allow for variations in the method of determining the sequential navigation (Frohlich, et al. 1997).

An alternative to instructional design tools with built-in knowledge about instruction (and instructional sequence) is to create tools that can learn new instructional strategies. As the instructional designer becomes familiar with different strategies, the design tool can have the strategies added to its repertoire and begin to incorporate them into the advice system. Id Expert™ is an adaptive instructional development tool that modifies the instructional strategies on-the-fly as the learner works with the system based on the learner's interactions with the system (Merrill 1997). While the system does try to modify the instructional strategies, its advice is limited to selections from its original set

of strategies. Id Expert™ creates instructional hypermedia using Instructional Transaction Theory (ITT) as the basis for modelling. ITT based models are good for sequential navigation but provide a limited set of data on which to base data model navigation.

Another system, MetaLinks, allows instructional designers to created hypermedia applications based on a hierarchical model of information (Murray, et al. 1998). MetaLinks doesn't use an explicit conceptual model, instead individual files are the atomic units of information and the organization of the information is assumed to be hierarchical. MetaLinks uses a depth first search to create instructional materials for learners and allows the instructional designer to tailor the information presentation through user-defined link types. One interesting feature of MetaLinks is the capacity for instructors to control the depth of traversal of the information hierarchy and for learners to request additional depth. In effect this allows the instructional plan to be modified dynamically (Murray, et al. 1999) although the strategy used to create the materials is always depth first through the hierarchy.

The designers of intelligent tutoring systems (ITS) take a slightly different approach to the creation of instructional hypermedia. The goal of most ITS designers is to automate the instruction for specific, goal-oriented tasks. Often, but not always, these tasks are behavioural in nature, such as instruction on the operation of electronic devices. As a result, ITS designers represent the instructional domain as a series of concepts that build up to an understanding of a final, goal concept.

One such system, the Dynamic Course Generator (DCG), builds a model of the domain and uses that model to automatically create tutorials for learners (Vassileva 1995). DCG represents the concept structure separately from the materials and the course is generated by applying algorithms to the concept structure. The algorithms for course generation also take into account the system's internal knowledge about instruction.

The DCG includes the notion of codifying knowledge about instruction to allow the rules to be used to generate instructional material. DCG uses discourse rules to constrain and define the final hypermedia tutorial. DCG provides a basic set of

discourse rules and a design that allows for additional rules, but the author notes that it is extremely difficult to systematically record knowledge about teaching, largely because teachers are not good at describing the teaching processes they use (Vassileva 1995).

2.5 Conclusion

Instructional design is similar enough to software design to allow the application of software engineering principles to the instructional design task. The similarity is even more apparent when the design task is the creation of instructional hypermedia, because the final product bears striking similarity to a software application. A software engineering approach to instructional design allows one to use common notation (the UML for instance) for the models representing the application. However, instructional designers are rarely also software professionals, so the development process cannot simply be the same as for software. Instead, the process for instructional design must incorporate the processes that instructors already use. Educators spend a significant portion of their design time planning the sequence in which material will be presented to learners. The difficulty in using a software engineering or hypermedia design metaphor for instructional design is that the sequencing of the presentation is not usually considered and never to the extent that is required for instructional design.

A design process for instruction must include a modelling representation for the sequencing of instructional materials. This sequence is in part determined by the concepts to be taught, in part determined by the characteristics of the learners, and in part determined by the purpose underlying the instruction. Instructional designers use their knowledge of instructional strategies and their own personal views on instructional theory to shape how they organize instruction. A tool that supports instructional design must have some mechanism for giving advice that is compatible with the views of the instructional designer and for incorporating new views into its advice schema.

Instructional hypermedia is time consuming and expensive to make, so most applications are designed to appeal to as broad an audience as possible. The hypermedia application is created to be a general as possible allowing it to be used for

many different purposes. Unfortunately, this approach often results in an application that is less useful than it could be for most purposes. A tool to support the development of instructional hypermedia must allow for the creation of specialized hypermedia applications from reusable information components. The same information should be useable in an application designed to provide a review as in an application that was intended to teach a specific new skill. This sort of reusability requires that the creation process be at least partially automated to reduce the time required to create new instructional applications. It also requires that the site structure and navigability be measurable and testable.

Instructional designers require software that supports the development of instructional hypermedia applications. The software must guide developers through a systematic design process and should make use of metrics to ensure a quality application. It must incorporate known good practices and solutions to common hypermedia design problems, since the users are instructional designers and not necessarily experts in hypermedia. The design environment must also provide support for different approaches to instruction and allow the designer to tailor the final product to different instructional goals. Many support environments exist, but none fulfill all of the above requirements. The following chapter presents an approach to developing instructional hypermedia applications that considers each of these issues.

Chapter Three

APHID: A Framework for OO Development of Instructional Hypermedia

Typically we think of the purpose for instruction as simply to enable learning about a particular topic. While the overall goal of instruction is to communicate the concepts within a certain domain to learners, there are secondary goals as well. These goals vary, but generally are specific to the type of knowledge the learners are intended to be acquiring and how best to communicate that knowledge. These secondary instructional goals lead to decisions such as when to provide an overview, a review, or a detailed understanding. Hypermedia applications that are intended to be instructional must reflect these different instructional purposes, as well as the structure of the actual material in their design and presentation.

An instructional hypermedia application that presents a review should have a different structure from one intended to provide an in-depth study of the same topic. It should contain more succinct information and the sequence of concepts should be arranged differently. The differences in structure can be described using concept maps, domain models, and a set of rules for sequencing presentation. More generally, these differences can be described using a patterns-based description to facilitate communication between designers and instructors. These patterns, along with an object-oriented analysis and design (OOAD) approach and a model that reflects the instruction as well as the content can be combined to create a structured repeatable design process for instructional hypermedia applications. The resulting development approach is called *Applied Patterns for Hypermedia Instructional Design (APHID).* APHID is an object-oriented approach to hypermedia design similar to OOHDM (Schwabe, et al. 1996). The differences are that APHID explicitly represents instructional components and sequences and that APHID uses pattern descriptions to ensure that both instructional

experts and software designers have the same understanding of the intended hypermedia. It allows designers to specify preferred sequences through the hypermedia application and to identify the instructional aspects of specific materials that will be used in the application. Any object-oriented process would suffice for the basic design methodology, as would any notation that supports object-oriented design. For this research a notation based on the UML has been developed as the vehicle to represent the designs for the hypermedia applications and general object-oriented methods are employed rather than one particular process.

APHID consists of software tools to support the creation of instructional hypermedia applications and a prescribed process for creating the applications. The process involves first defining the domain of instruction with a concept map, then creating the elements that will make up the pages of the hypermedia application. The software tool contains functionality supporting both of these steps. The hypermedia application is generated from the concept map and page elements using configurable constraints and rules that are part of the software environment. APHID creates hypermedia to be used as an individualized instructional aid rather than hypermedia that is intended to do the actual teaching. The remainder of this chapter details the APHID development process.

3.1 Creating Instructional Hypermedia

The APHID approach to instructional hypermedia application design is somewhat different from that usually taken when constructing instructional materials. Typically the instructional designer works from a curriculum and creates the materials to follow the pre-set curriculum. In our approach, the instructional designer first creates the map of the concept space, then creates the data elements that are the instantiations of those concepts. The curriculum or sequence is created semi-automatically later in the development process. For instance, if the instructional topic was *data structures*, the concept map would have *data structures* as the root of a graph and concepts like *stacks, trees,* and *arrays* as child concepts related by abstraction relationships. Each concept could have any number of data elements associated. Perhaps the *data structures* concept has data elements that are paragraphs describing data structures in general. The *stacks*

concept may have a data element that is a simulation allowing the learner to observe how a stack works.

For instruction, consideration of semantic characteristics as well as static characteristics of the data is required. In addition to understanding which objects interact to create the application, it is also important to understand what the possible sequences of presentation are (from a learner's perspective) and to determine which of those are appropriate for any given instructional purpose. These sequences can be represented as algorithms describing a traversal of the graph representing the concept map. As each node in the graph is traversed, it becomes the next part in the instructional sequence.

The principles associated with software patterns can be applied to the design and construction of instructional hypermedia applications. The sequence of instruction, with respect to the concept map, and size characteristics of the application can be described in terms of patterns. Each pattern, in this case, would represent a known 'good' solution to the problem of presenting the next important piece of information for a particular user, given a particular instructional goal. Describing the sequences as patterns is not requisite for the successful functioning of the development approach, but it is useful to communicate what APHID does, both to instructors and to software designers.

3.2 Instructional Hypermedia

Instructional hypermedia differs from other hypermedia because the instructor anticipates and perhaps even imposes the goal of the learner. From a modelling perspective, this both simplifies and complicates the modelling task. Instead of modelling a general purpose hypermedia application to fit a variety of user purposes, assumptions can be made about what the user wants to do or ought to do (from the instructors perspective). This is possible because the learner may be given a specific task to accomplish rather than always being allowed complete freedom to self-select a goal. The designer must understand how the different instructional goals change the presentation of the material. Nonetheless, predictions can be made about the sequencing and presentation of the information in an instructional web site, based on the learning goal imposed externally by the instructor.

Recall the example of the conceptual model describing *data structures*. The model will include concepts to describe the operations possible on the different types of data structures as well as concepts describing the use of the various data structures. If the instructional goal is to present a short introduction to data structures, it is unlikely that the learner requires detailed information about how the *pop* algorithm for a stack works. An instructor would just skip those parts of the concept map that were not relevant to the present learning goal.

It is the same for the development of hypermedia applications. The presence of a clear learning goal often allows the designer to eliminate portions of the conceptual model (which may be quite large) from the model of the hypermedia system. Any parts of the conceptual model that are not relevant to the learning goal are simply ignored for the specific hypermedia application under consideration. This does not imply that those concepts are eliminated for all applications constructed for the particular domain of instruction, just for the one specific learning goal.

3.3 Models for Instructional Hypermedia Analysis and Design

The design of instructional hypermedia is similar to construction of any other hypermedia application. It requires similar sorts of modelling, similar sorts of analysis, and the same kind of presentation considerations. The difference is that instructional hypermedia requires two sorts of navigation: navigation that follows from the model of the domain, and navigation that follows from the model for the sequencing of instruction (Frohlich & Nejdl 1998). These two different navigational models can be determined if the design of an instructional application includes both a domain model that represents the 'types' of the data and a model of the semantics associated with the instructional domain. The APHID environment utilizes both a domain model and a semantic model.

The Basic Computer Science Concepts (BCSC) tutorials (Tokarchuk 1998) were created to provide on-line instruction for Computer Science students who were lacking an understanding of basic concepts. Each tutorial covers one basic concept, such as trees, sorting or stacks. The tutorials were constructed by hand, but the concepts and data

elements from them have been used as the test materials for this research. The remainder of this section contains examples taken from the BCSC tutorial about stacks.

3.3.1 Domain Model

Within APHID, the domain model for an instructional hypermedia application consists of the model of the types of instructional materials that make up the application. These types are called *instructional classes*. In a sense, the domain model for any specific instructional hypermedia application is a subset of the model that represents all possible instructional classes. The second part of the data model is meta-information about the interacting objects that make up the hypermedia application. Each such object is called a *data element*. A data element is, in some senses, an instance of an instructional class. Each data element represents a discrete unit of content to be delivered, for example, a data element about *stacks* could be the algorithm for pushing an item onto a stack. The meta-information about these elements allows APHID to record the relationships between elements and the concepts as well as information about how the elements relate to one another. The addition of meta-information about each element to the data model results in a model that is quite specific, but still more general than the finished hypermedia application.

3.3.1.1 Instructional Classes

Instructional classes are represented as a class diagram in the common object-oriented sense. The difference between instructional classes and most other domain models for hypermedia is that the instructional classes represent the 'meta-domain' of providing instruction rather than the content domain. In the case of instructional hypermedia it is these meta-concepts that provide the structure to the hypermedia application.

For instance, the Art Museum used as an example by Schwabe et al. (1996) contains classes such as *Person*, *Artwork* and *Reference*. These classes represent domain elements that occur frequently and share common characteristics as data classes. A hypermedia application created using OOHDM is organized using information about the characteristics of the data.

In this research the domain is instruction (rather than an art reference manual). The organization of instruction depends equally on the characteristics of the content and on the characteristics of the instruction about the content. The classes of concern in this domain are the different types of information that can be conveyed in an instructional setting. So, for an instructional application the classes can be represented as things like *Instructions*, *Example, Exercise* and *Definition*. Using these classes, an instructional application can be developed with attention to the characteristics of the data. For example, instructions are a sequence of steps, while a definition is a concise statement about the nature of the concept. Each of these types of data must be handled differently when presenting them to learners. For the remainder of this document, such classes will be called *instructional classes*. Figure 3-1 shows a few sample instructional classes

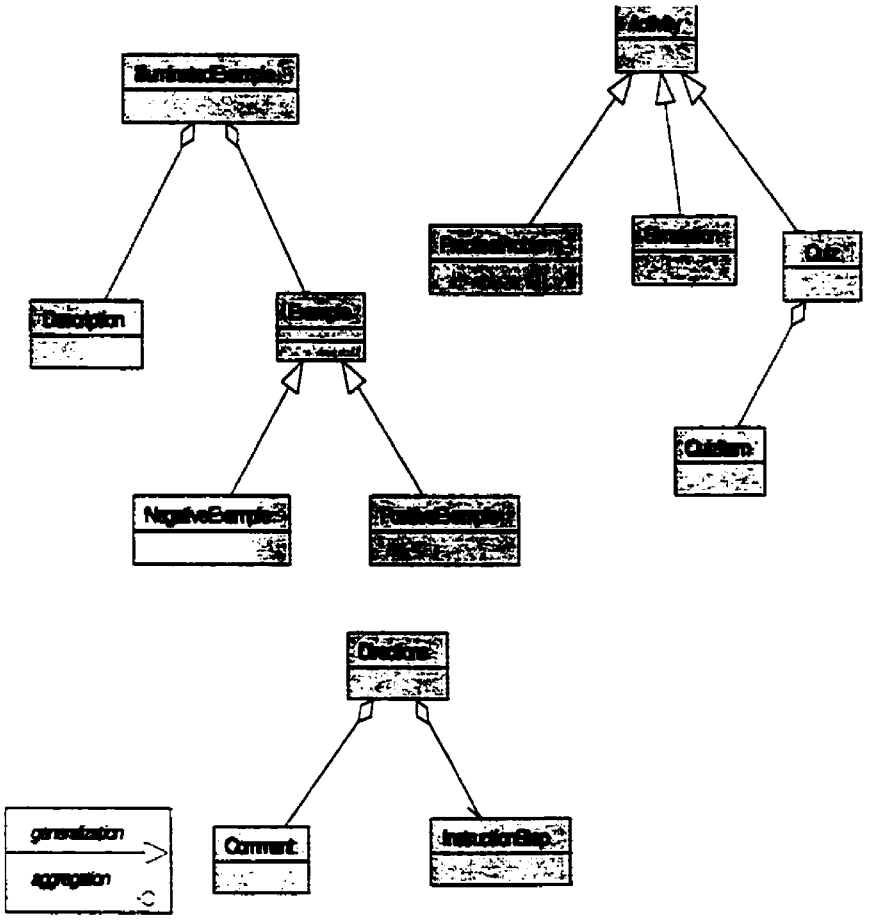


Figure 3-1 Sample Instructional Classes

complete with aggregation and generalization linkages.

3.3.1.2 Data Elements

The creation of a hypermedia application with APHID depends on the definition of the data elements. A data element consists of the data (text, video, pictures) that the learner will view along with additional meta-data that are used by APHID to create the hypermedia application. Data elements are described using a small set of meta-data that describes the data's relationship to other elements of data in the application. The meta-data used is similar to that identified by (Instructional Management Systems (IMS) 1999). The data element contains both the information that will be displayed to the user of the hypermedia application (the learner), and the information about how that element fits into the organization of the application.

Data elements comprise the information that the learner will actually see as well as information about how that data element should be presented. Each data element is indexed by a 6-tuple containing the meta-data for the element: [**instructional class, concept, media, level, context, importance**]. Each unique 6-tuple can be considered a special kind of 'class' which most often has only one instantiation in the hypermedia application, but could be instantiated more than once. For instance, consider a 6-tuple representing a quiz item data element about a single concept. Such a tuple could be instantiated many times because there could be many quiz items that have the same concept, media, level, context and importance. The only difference would be the actual data that learners see when the hypermedia application is created.

Of the characteristics represented in the 6-tuple, **media**, **level** and **importance** are simple lists that have three or four possible values. **Instructional class** represents the class (as shown in Figure 3-1) to which the data element belongs. **Concept** indicates with which concept, from an instructional viewpoint, a particular data element is associated. The **context** attribute supplies the algorithms with an understanding of the context in which a sub concept is being described. For instance, the concept a data element is associated with might be the main topic of discussion or it might be a sub-topic. Some data elements will be more appropriate for use in a secondary context, while others will be more appropriate for use when the element's concept is the primary

focus of the instructional materials. The attributes of data elements, along with their possible values are detailed in Figure 3-2.

Category	**Possible Values**
Iclass (Resource Type)	Any of the instructional classes
Media	Any valid media type (uses mime types)
Context	(Primary, secondary)
Concept	Any concept from the concept map
Importance	(Critical, explanatory, enrichment)
Level	(Novice, intermediate, advanced)

Figure 3-2 Data Element Attributes

The instructional class and the media type determine the permissible methods of displaying the data element. The level and importance determine which elements become part of the hypermedia applications and the associated concepts determine the permissible placement of the data element within the hypermedia application as a whole.

Once all the data elements are in place, the curriculum, the instructional plan and the instructional materials can all be created semi-automatically. Those traditional teaching documents (i.e. curriculum, instructional plan, etc.) are a view (in the database sense) on the data elements. The view is shaped by the knowledge contained in the concept map and by the enactment of instructional patterns that guide the creation process (Section 3.4).

3.3.2 Semantic Model

Instructional classes do not provide information about the knowledge that is being communicated. The individual pieces of knowledge (data elements) are instances of one (or more) of the instructional classes, but the instructional class gives no understanding of how that data element relates semantically to any other data element. Missing is the

portion of instructional design that selects the sequence of the concepts. Knowing that a data element is a quiz question rather than a definition is important, but doesn't give enough information to determine if it should appear before or after some other data element. Sequencing of information, from an instructional perspective, is dependent on the structure of the knowledge being taught, rather than on the static structure of the data elements. The semantic structure can be represented in a concept map. This knowledge about the instructional domain can be used to guide the sequencing and organization of the hypermedia application. We call this model the *semantic model.* Used for hypermedia application design, a concept map allows instructional hypermedia designers to create a model of the semantics of the data that can then be used to design the instructional navigation.

Consider a traditional classroom-based instructional process. The educator selects concepts to present to learners and chooses the format, the detail and the order of the concepts. The sequence in which concepts are presented, and the selection of one concept over another (by the instructor) is similar to navigating through a concept map for the domain of instruction. If the concept map is considered to be a directed graph, the path through the graph represents the instructor's choices for material. For any particular instructional purpose, say providing a review, the sequence (or pattern) through the concept map should remain more or less the same, regardless of the problem domain.

To be useful to hypermedia designers, the concept map must be expressed in a consistent format, regardless of the instructional domain. For instance, a concept map will represent certain types of associations within any domain, perhaps an ordering relationship to indicate that one concept is prerequisite knowledge for a second concept. It is desirable that the relationship types remain constant across instructional domains, to allow designers to make decisions based on the structure of the concept map without consideration for the specific instructional material. This requirement necessitates a specification for a generic instructional concept map.

The construction of a detailed concept map for the domain of instruction is the first step in the APHID approach to instructional hypermedia development. The structure of the

map, along with an understanding of the purpose of the instruction governs how the hypermedia application will be assembled. The key is the associations that concepts have with other concepts because the algorithms that guide the creation of the hypermedia application make use of the associations.

An example concept map for the Stacks tutorial appears in Figure 3-3. Four kinds of associations appear: *aggregation, generalization, prerequisite* and *uses*. Aggregation and generalization have the same application and meaning as for object-oriented software design, *part-of* and *is-a* respectively. They are represented in diagrams using the UML notation for aggregation and generalization. *Prerequisite* and *uses* links have no direct counterparts in OO analysis and design.

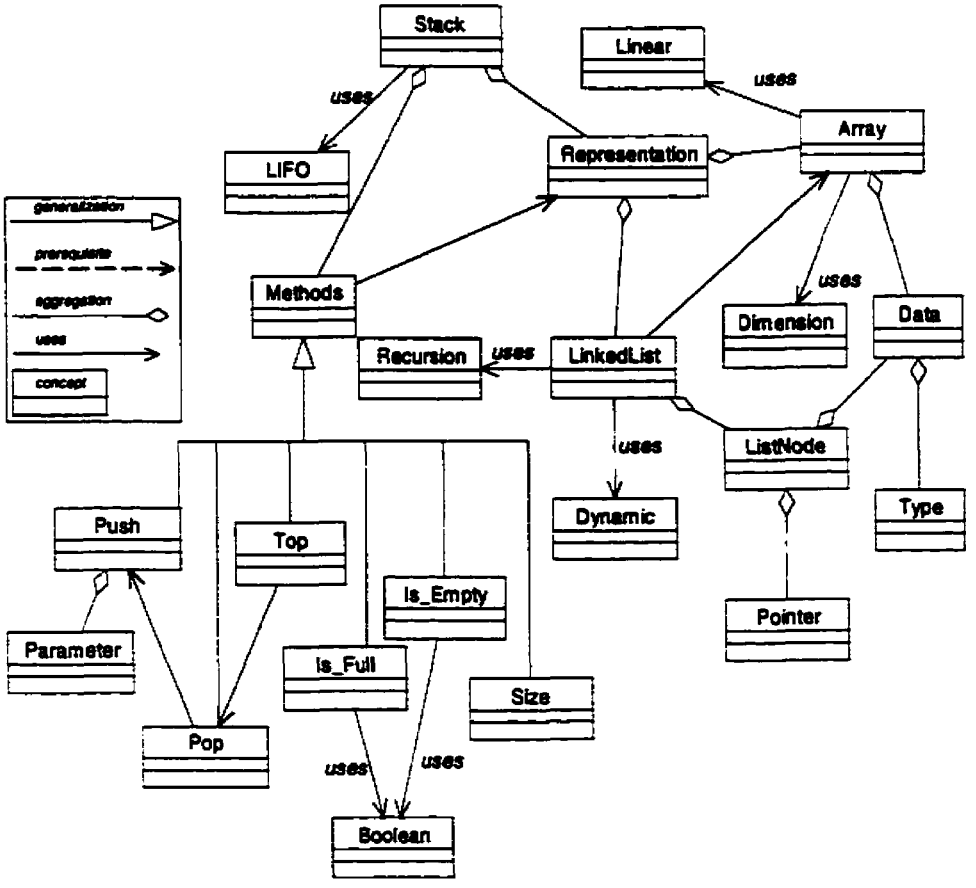


Figure 3-3 Concept Map for Stacks Tutorial

Prerequisite links between concepts indicate that an understanding of Concept A is dependent upon some information in Concept B. The UML notation for dependency is used to represent this association, although the definition of prerequisite is not exactly the definition of dependency. Prerequisite links are allowed only between 'siblings' or concepts that have the same immediate parent and are connected to that parent by the same type of link (i.e. aggregation). For example, there is a prerequisite link between the concepts of Pop and Top in Figure 3-3. This link indicates that the concept of Pop should be taught before the concept of Top.

Uses links define an association between concepts indicating that one concept requires knowledge of the other for complete understanding. They are similar to the prerequisite relation, but not as strong and may exist between concepts that are not peers. A *uses* link indicates that one of the two concepts in the relation contains information that is background or prerequisite information for the other concept. But, the relationship is weak enough that it is not crucial that the background information be understood in order to understand the second concept, in other words it is acceptable to teach the dependent concept without teaching the background concept for some learning goals.

The concept map defines how the topics associated with the domain of instruction will be presented to the learners. It represents the instructor's understanding of the domain of instruction. The root of the concept map (stack in the example) represents the most general topic of instruction. Most often, instruction begins at the general concept and proceeds through the sub-concepts in some order. The sequence is determined by the instructional goal, but the sequence will only be accurate if the concept map is constructed correctly. Its construction requires extensive knowledge about the domain of instruction and a clear understanding of how that knowledge is structured.

3.3.3 Navigation Model

The navigation model for an instructional hypermedia system must incorporate information from both the domain model and the semantic model as well as the desired instructional purpose (which implies a particular sequence through the material). Both data model and sequential navigational possibilities must be represented. This includes

navigation between the different nodes (pages) in the hypermedia application and navigation from data element to data element within each node. A node in the navigation model can be thought of as a class or data structure with several members. Due to the difficulty of graphically representing this complex information for an entire hypermedia application, the navigation model is not represented as a diagram in the APHID approach. Instead it is inferred (algorithmically) from the domain and semantic models using instructional and site design patterns, and is presented to the user as the finished application.

Figure 3-4 shows the portion of the APHID conceptual model that details the structure of the navigational model. The many-many associations (indicated with 0..* notations on the relationships) make diagramming the navigational model difficult, but because it

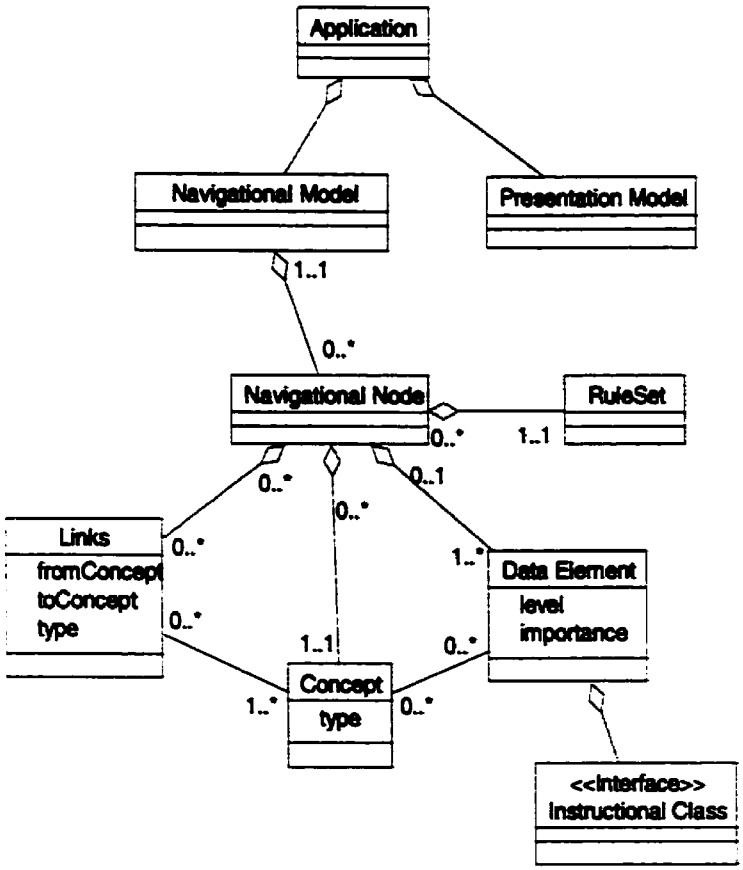


Figure 3-4 Partial Conceptual Model for APHID

is quick and easy to modify and regenerate applications, the lack of an explicit user-diagram for the navigation model is normally not problematic.

APHID uses the semantic model and the instructional goal to plan a sequence for the instruction. The sequence could be completely linear, if the instructional goal was something that was compatible with a linear presentation, but is more likely to have several choice points for students. Once the sequence is established, each concept is examined and the appropriate data elements are selected using the constraints on ability and depth of coverage provided by the designer. After data elements are selected, the navigational model is completed giving priority to the sequential navigation derived from the semantic model, and adding data model navigation wherever appropriate. User interface constraints and patterns guide the addition of data model navigation.

3.3.4 Presentation Model

Each instructional class can be presented in a variety of ways. The presentation model for an instructional class describes how the information will be presented given the constraints imposed by the application developer. For example, one data element that represents an activity might be best presented as a series of pages due to its length or complexity, while another data element describing an activity might be best presented on the same page as the explanation. The creation of a presentation model must take into consideration all the characteristics of both the instructional classes and the data elements. The presentation model for APHID will consist of a set of templates, described as patterns, defining possible presentations for different instructional classes. For example, one kind of instructional class is the quiz. In some situations the instructor may wish for a quiz to be presented to the learner a single quiz item at a time. In other situations the quiz should be presented as a single unit. Presentation patterns describe the different presentation possibilities that APHID supplies and allow the designer to make informed decisions about which presentation template to apply to a particular data element. The templates and patterns are defined using common user-interface design principles (Nielsen 1993).

3.4 Patterns for Instructional Hypermedia

Two types of patterns are used to guide the construction process: *presentation patterns* and *organizational patterns*. Presentation patterns are patterns that guide the construction of the hypermedia based on structural characteristics of the data and known user interface design principles. Presentation patterns are similar to the patterns identified by previous researchers (Garrido, et al. 1997; Manns 1998). Organizational patterns guide the construction of the hypermedia from an instructional viewpoint. It is the organizational patterns that give guidance about which materials should be presented first, and how extensively each topic should be explained.

Patterns to guide the creation of instructional hypermedia are described within the context of the concept map for the domain (which indicates the characteristics of the data) and the instructional goal of the hypermedia developer. The patterns describe approaches to creating hypermedia that result in applications that are well constructed and educationally worthwhile. These instructional patterns work on the assumption that the structure of content has common characteristics across the potential domains of instruction and across various learners. Patterns for both hypermedia design and pedagogical design can be used to guide the construction of the finished application. In the case of APHID the pattern descriptions also include algorithms that represent the instructional pattern applied to a concept map. APHID patterns are useful both to the educator and to the software specialist.

3.4.1 Presentation Patterns

When considering a single concept, the design and construction of the hypermedia application is relatively straightforward. Even if the concept has several 'pages' worth of information, design principles and patterns can be followed to produce a well-designed hypermedia application based on the characteristics of the data elements and the instructional class of the data. Any design process can be used successfully to create well-constructed pages with predictable hyperlinks between the types of data. Presentation patterns consider the presentation model and the navigation model when proposing recommendations for the construction of a hypermedia page.

Instructional hypermedia pages can be constructed algorithmically by using presentation patterns to guide the visual appearance of the pages and rules (constraints) to govern the quantity of information and hyperlinks on the page. The rules function in a similar fashion to golden rules, (Nanard & Nanard 1998) which specify guidelines for making choices about structural elements and presentation styles. Although the hypermedia designer could control each attribute of the presentation directly, presentation patterns guide the designer through decisions such as the combination of instructional classes to create hypermedia pages and the placement of hyperlinks within the data elements. Presentation patterns within APHID are described with default settings to be used to automatically generate hypermedia pages without necessitating designer intervention.

3.4.2 Instructional Patterns

Instructional patterns are the organizational patterns for instructional hypermedia. They consider the semantic model and the domain model along with the instructional purpose for the hypermedia application. They give suggested solutions for decisions such as: which concept to discuss next, and which concepts to explore more fully than others. Again, the instructional designer can make all of these decisions unaided, but the instructional patterns provide guidance for those who wish it. They represent instructional decisions that are known to have worked previously in a particular situation and they enable automation.

The goal of the instructor (or the instructional designer) determines which organizational pattern is used to guide the construction of the hypermedia application. For instance, an instructor may want to give learners a detailed treatment of a topic, in which case each concept in the concept map must be examined thoroughly with prerequisite concepts being examined first. On the other hand, it may be the instructor's intent to provide an overview only, in which case each concept should be covered, but only superficially, and some unimportant concepts may be left out entirely. If remediation is the goal of instruction, a review of prerequisite concepts is in order followed by an in-depth study of the concept to be remediated. Other possible instructional goals include review, preview and enrichment, each with their own pattern

of instruction. Like presentation patterns, instructional patterns can be described for communication with instructional designers and can be represented with algorithms to facilitate automation.

Four instructional patterns have been defined as part of this research. Two of them, Depth First Instruction and Spiral Instruction, are presented here as examples. These patterns contain algorithms that model the process an instructor might use when planning instruction of this sort. The algorithms can be implemented, using a concept map, to allow semi-automated construction of the hypermedia application. APHID uses the constraints associated with a project to select a sequence of topics from the concept map and to create a hypermedia application. Unless constraints are unsatisfiable, APHID will create the entire application unaided. If the constraints are complex, or if the designer wishes to have more control over the development process, APHID allows for human intervention during the development process.

Depth First Instruction (DFI) is a pattern of instruction used to teach the target concept by teaching each sub-topic in turn, in its entirety. The DFI pattern is most suitable when learners are missing a large amount of background material or when the concepts being presented are only loosely related to one another. The Depth First Instruction Pattern is given in Figure 3-5. Using Depth First Instruction to create a set of instructional materials requires a concept map and the data elements associated with it. The pattern indicates that a depth first traversal of the concept map is performed, starting at the root concept. The types of associations between concepts (aggregation, generalization, and uses) are handled individually. At each concept node, the data elements for that node are retrieved and assembled into hypermedia documents using a separate set of patterns and rules that describe how instruction for a single concept should be presented. Using APHID, the concept map, and the structure of the data elements, the hypermedia application can be assembled automatically.

Spiral Instruction (see Figure 3-6) is a pattern used when the instructor wishes to gradually expose the learners to each concept. Each concept in the map is presented many times, each time in increasing detail. Each presentation builds upon the knowledge presented in the previous segments. A Spiral Instruction pattern is indicated

Depth First Instruction(DFI) (Organizational)

Intent

Each concept in a concept map should be explored once in depth.

Motivation

An instructor who wishes to provide an in depth coverage of a set of concepts generally teaches each sub-concept completely before moving on to the next sub-concept. This pattern suggests a method for ensuring that all of the sub-concepts that comprise a topic concept are covered in depth to ensure a comprehensive understanding of the target topic.

Applicability

Use the Depth First Instruction pattern when:

- Students require instruction for the entire target domain, including prerequisites.
- Students are mature enough to understand the concept, but lack background knowledge.
- Students are unlikely to require long periods of time to digest sub-concept material before moving on to a new concept.

Consequences

The application of this pattern results in a curriculum that suggests a linear path through the set of concepts.

Implementation

Depth First Instruction is limited by the quantity of data elements. Concepts for which there are few data elements may not be examined in depth, even when using this pattern.

The organization of the concept map has a major effect upon the final application. The algorithm makes use of the different types of links between concepts to order the information within the hypermedia. *Uses* links are processed first, followed by aggregation links then abstraction links. Prerequisite links take precedence over all other link types.

The use of an incorrectly organized concept map results in a poorly constructed application.

Algorithm

The algorithm for creating a curriculum and instructional materials using Depth First Instruction is given below. It is essentially a Depth First Search through the concept map with some consideration for the different types of links.

Process the current concept

1) *Create the instructional materials for the current concept using presentation patterns and constraints for the project.*
2) *For all child concepts.*
   a) *Sort the concepts according to link type and prerequisite information*
   b) *Recurse to process each concept in order*

Figure 3-5 Depth First Instruction Pattern

when learners require a gradual introduction to each concept in the map or when the concepts are strongly associated with one another.

Spiral Instruction (Organizational)

Intent

Each concept in the concept map should be explored several times, each time in more detail.

Motivation

An instructor who wishes to provide complete coverage of the target concept in an incremental fashion should use this pattern. This pattern suggests a method for ensuring that all of the sub-concepts that comprise a topic concept are covered in depth while also ensuring that learners are not overwhelmed by too much information at any one stage.

Applicability

Use the Spiral Instruction pattern when:

- Students require instruction for the entire target domain, including prerequisites.
- Students need time to understand preliminary material before going on to more advanced material.
- Concepts are highly related and are best presented in parallel.

Consequences

The application of this pattern results in a curriculum that spirals through the set of concepts.

Implementation

Spiral Instruction is limited by the quantity of data elements. Concepts for which there are few data elements may not be examined in depth, even when using this pattern. Spiral Instruction is most effective when a concept has data elements at a variety of levels.

The organization of the concept map has a major effect upon the final application. The algorithm makes use of the different types of links between concepts to order the information within the hypermedia. The use of an incorrectly organized concept results in a poorly constructed application.

Algorithm

The algorithm for creating a curriculum and instructional materials based on Spiral Instruction is given below. It is most similar to an Iterative Deepening search through the concept map. The Depth variable is initially set to one.

Create the instructional materials for the current concept using presentation design patterns.

If current concept has been covered in depth, create a review statement,

Else create an in-depth treatment of concept.

In order (as suggested by link types and prerequisites),

create the instructional materials for each child and the child's children until Depth is reached.

Increment Depth and repeat.

Figure 3-6 Spiral Instruction Pattern

3.5 Using APHID

The use of APHID consists of three distinct steps. First the designer creates a concept map and several data elements. Next the designer specifies constraints by setting various system parameters or by choosing a built-in pattern which assigns default values to the parameters. Finally, the hypermedia application is automatically constructed by APHID using the map, elements and constraints provided by the designer. This section gives an overview of the hypermedia development process using APHID.

An instructional designer working with APHID first uses the concept mapping environment to define the concepts for the domain of instruction. This definition involves naming each concept, identifying whether it is a key concept in the domain or a less important concept and providing the associations between the various concepts.

After the concept map is defined, the data elements must be detailed. The actual information that learners see must be created and the meta-information about each element must be recorded in APHID. Meta-data about the element indicate which concept(s) the element belongs to, how difficult the element is, how the element should be displayed, etc. After the concept map and data elements are created, the designer must choose the type of application desired (e.g., review, introduction, in-depth instruction or reference work), make adjustments to the default settings if desired. After this the work of the instructional designer is complete. The APHID environment can then create the hypermedia application. APHID maintains a set of constraints for the hypermedia application and is governed by those constraints while creating the hypermedia application.

For instance, suppose the designer wishes to create an application that allows learners to study the given topics in a domain in depth. This is the sort of approach to instruction that might be taken when introducing a new domain to learners. APHID first uses the Depth First Instruction pattern to move from concept to concept in the concept map. As it reaches a new concept, the data elements for that concept are retrieved and examined.

Suppose the instructional designer has indicated that only the simpler data elements should be used for this application. APHID then selects only the data elements with a

suitable level attribute. The set of elements is examined for number, length, content type and presentation suggestions. APHID decides, based on the presentation patterns associated with each element and on internal patterns suggesting basic hypermedia design, how the data elements should be organized on the pages, how many pages should be created for the concept, which other pages in the application should be linked to, and where the hyperlinks should appear. Hyperlinks are created based on the associations in the concept map and on associations within the instructional classes. Hyperlinks are represented in a navigation bar that is on the perimeter of the page. Presently no provision for including in-text hyperlinks is made.

The approach that APHID uses is somewhat similar to the navigational contexts proposed by Schwabe et al. (1996), the algorithms used in ITT (Merrill 1996), and to the discourse rules used by the DCG (Vassileva 1997). It combines the approaches of these research efforts, but hides the complexities of specifying the rules from the hypermedia developer.

In summary, APHID supports the instructional designer in building well-designed, sound hypermedia applications to use as teaching aids. An environment for creating the necessary models, and patterns to simplify and guide design decisions scaffold the instructional design process. This chapter has given an overview of the approach used in APHID. The following chapter provides details about the software environment and the methods and algorithms used to create hypermedia applications.

Chapter Four

Detailed Design and Implementation Details

APHID is a hypermedia development environment for instructional designers. It facilitates the creation of instructional hypermedia applications in the form of web sites without requiring knowledge of HTML on the part of the instructional designer. The APHID environment helps the instructional designer define the domain of instruction using concept maps, create materials for the application through a data element template and build a working hypermedia application using the application construction tool.

APHID provides a support environment in which instructors can design and create well-built hypermedia applications with a minimum expenditure of time. From the instructional designer's point of view, the creation of a hypermedia application consists of first using the software to create a concept map that represents the domain of instruction, then creating the actual materials that learners will see in the finished application. The final modelling step is to use the software to document relationships between the concepts on the map and the data that learners will see. After that, building an application is as simple as setting two parameters (type of application and level of user) and clicking a button. The APHID software decides which concepts to include, which data elements to include, which hyperlinks to create between pages, and how to display the selected items based on the models and the supplied parameters.

This chapter focuses on the detailed design and prototype implementation of the APHID development environment. Throughout the chapter reference is made to an example application, created with APHID, designed to teach the computer science notion of stacks. This example uses much of the information contained in the Basic Computer Science Concepts (BCSC) tutorial (Tokarchuk 1998) about stacks.

The first section of the chapter explains the intended uses of the system. The second section details how APHID uses the various models associated with instructional hypermedia and explains how those models are represented. The third section of this chapter gives details about how a hypermedia application is constructed using the models developed by the instructional designer. This section explains some of the algorithms used by the APHID process as well as examining the role of XML in the application-creation process. The chapter concludes with a look at parts of the example application about stacks.

4.1 Use Cases and Package Definition

The APHID software is designed to allow instructional designers to create hypermedia applications. It is also designed to facilitate the inclusion of new instructional strategies (in the form of organizational patterns) and instructional classes. The administrative tool for adding such information to APHID has not been implemented, so although provision is made in the application design for adding new instructional strategies, presently the only method of adding them is by modifying the source code.

Figure 4-1 shows the use case diagram for the entire APHID environment. For diagram simplicity the use cases for edit and add have been represented as one use case. The only difference between editing an application and adding one is the loading of an existing application rather than creating a new one, so no functionality is hidden by merging the use cases. Only those use cases associated with the *Designer* have been implemented in the prototype software.

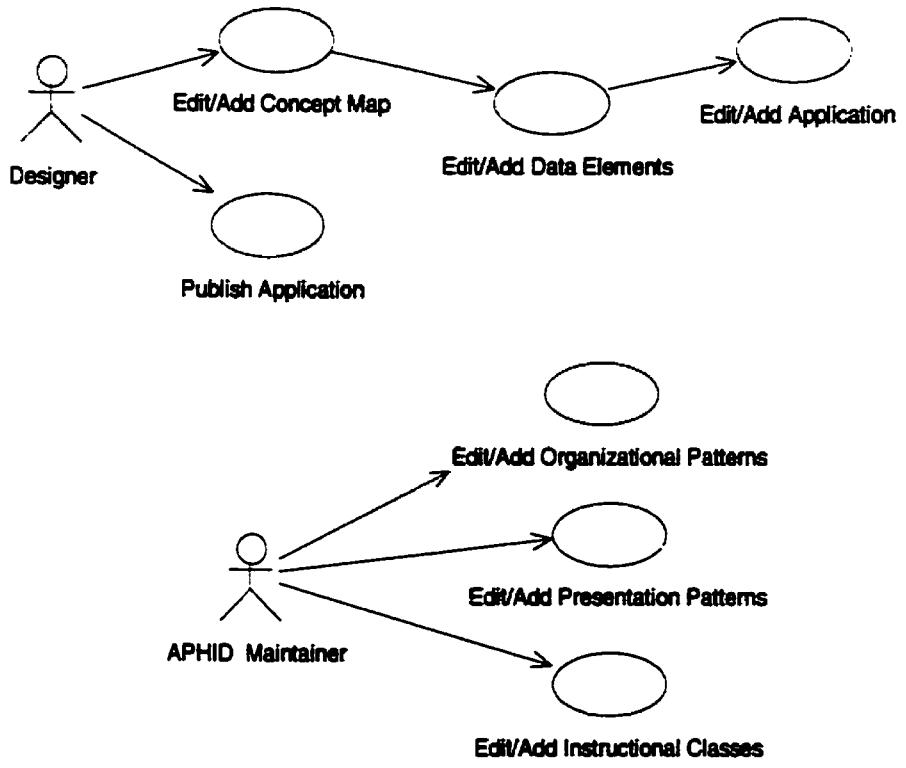


Figure 4-1 Use Case Diagram for APHID

The intention of most of the use cases is fairly self-explanatory; their function is to allow editing and addition of the materials required for building the hypermedia application. The high-level description for the main portion of each of the four use cases is given in Figure 4-2. The *publish application* use case is where most of the computing is done within APHID.

Use Case: **Edit Concept Map (add concepts)** Actors: Designer Description: This use case begins when the Designer chooses to add concepts or relations to a concepts map. The Designer adds a concept and selects the correct type for the concept. The Designer labels the concept with a name. The Designer adds associations to the graphical representation of the concept to indicate how it relates to the other concepts on the map.
Use Case: **Add Application** Actors: Designer Description: This use case begins when the Designer has a concept map completed and sufficient data elements defined to populate a hypermedia application. The designer indicates a desire to add a new application. The designer selects the purpose for the application from a list of purposes and chooses an appropriate instructional strategy. The designer also indicates the ability level of the target audience. The designer then assigns a concept map to the application and gives the application a name to identify it.
Use Case: **Add Data Elements** Actors: Designer Description: This use case begins when the Designer chooses to add a new data element to the current application. The Designer selects the type, level and importance of the data element. The Designer supplies the data that will be displayed to the user of the application. The Designer may, at this point, associate the element with one or more concepts although this step can be left until later. This use case has the precondition that the application be previously defined by the designer.
Use Case: **Publish Application** Actors: Designer Description: This use case begins when the Designer selects an application to publish from a list of completed applications. If desired, the Designer modifies the default application settings to further customize the finished application. Then the APHID environment uses the information about the application to generate a set of HTML pages that conform to the attributes of the application (ability level, instructional strategy, etc.). This use case has the precondition that the application to be published be previously defined by the designer.

Figure 4-2 Use Case Descriptions for APHID

The software functionality is accomplished via three main packages of code. A package diagram indicating these packages and their visibility is given in Figure 4-3. The *concept and element definition* and *patterns and instructional strategies* packages represent the domain objects and contain service objects to manage the data storage and manipulation. The *application definition* package represents the logic necessary to create a hypermedia application from the data definitions and the constraints associated with a particular instructional strategy. The *APHID maintenance* package provides means to extend the APHID environment with new patterns, instructional classes and instructional strategies. Each package is designed using a layered architecture approach to allow separation of the user-interface from the logic of the application. All packages except the APHID maintenance package have been implemented in the prototype software.

The APHID software first requires the instructional designer to create a model of the topic of instruction. This model is represented with objects from the concept and element definition package. The desired characteristics of the final site are selected and the attributes are represented using objects from the patterns and instructional strategies package. The model and the patterns are used by objects in the application definition package to build the hypermedia application.

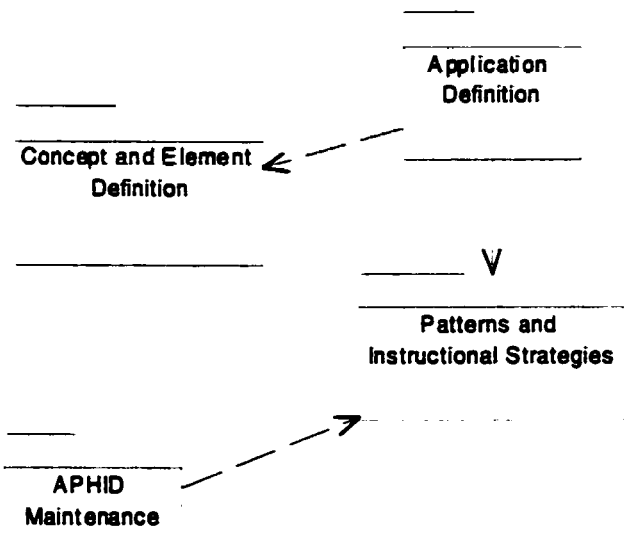


Figure 4-3 Package Diagram for APHID Prototype

4.2 Modelling

Hypermedia development with APHID relies heavily on the design and definition of three different types of models: a data model, a semantic model and a navigational model. The data model defines the instructional classes and the data elements associated with the hypermedia application. The semantic model defines the concepts and their relationships. The navigational model is created at run-time and defines the final structure of the hypermedia application after constraints have been satisfied. The navigation model is described in Section 4.3.1. The remainder of this section provides details about the data and semantic models, including a brief introduction to the eXtensible Markup Language (XML) (XML Working Group 1998).

4.2.1 Extensible Markup Language

The eXtensible Markup Language (XML) is a subset of the Standard General Markup Language (SGML) designed to allow users to define application-specific tags (unlike HTML which has a fixed set of tags) (Bos 1999). XML is appropriate for the definition of data elements within APHID precisely because it is extensible and allows the

definition of instruction-specific markup tags. In many ways it is similar to the schemas used to describe relational tables in a data base in that it illustrates the components that make up a piece of data and how those components are combined to create composite data items. XML is more powerful when combined with supporting technologies to produce dynamic documents from a single set of marked up data. In essence the XML portion of the document set is the database and the supporting documents describe which parts of the data should be rendered for the user and how those portions should appear.

XML is a set of rules for creating tags used for marking up text. HTML is also a markup language that focuses on the presentation of the text it marks up. Well-formed HTML (with both opening and closing tags for every element) can be treated as an XML document. Common tags in HTML denote heading size (H1, H2) and text appearance (I, B). HTML tags also delineate the function of the text within the presentation, such as whether or not a piece of text is part of a list. XML tags on the other hand are commonly written to explain the semantics of the text rather than the presentation. Often an XML schema can be translated directly into a schema for a relational or object database. The designer of the XML document type may express the data-model either as a Document Type Definition (DTD) which is a grammar-like description or as an X-Schema, which is more like a description of a relational database (North 1999). Presently the DTDs are more common and have more software support. APHID document types are expressed as DTDs.

APHID uses XML to describe the educational semantics of the data. In other words, the XML is used to identify the parts of the data associated with the different educational components that make up an instructional web site (instructional classes). Example of the document type definitions used in APHID are given in the next section.

4.2.2 Instructional Classes

The data model used in the APHID development process describes the domain of instructional hypermedia rather than the topic of instruction. This model can be represented in any object-oriented modelling language, such as UML, where each

```
<!ELEMENT description (#PCDATA)>
<!ATTLIST description
 type (intro|summary|conclusion|explanation|preformatted) "explanation">

<!ELEMENT instructions (description+, instruction_step+)>
<!ELEMENT instruction_step (description? |action)>
 <!ELEMENT action (#PCDATA)>
```

Figure 4-4 Example Document Type Definition

different type of instruction is represented as a class. The model illustrates the different instructional classes and their relationships with one another. Instructional classes have attributes and, in one sense, they have methods. The methods define how instances of the class are presented to the final user of the hypermedia application. Within the prototype software, these classes are represented by data type definitions (DTDs) for the eXtensible Markup Language (XML). The methods are represented as eXensible Style Language Transformations (XSLT) (XSL Transformations (XSLT) 1.0 1999). Each instructional class has a separate definition that describes the content of the class and how it can be combined to create more general classes.

APHID uses these definitions to guide the creation and storage of data elements as the instructional designer fleshes out the hypermedia application design. The document types for all the instructional classes implemented in the prototype software are given in Appendix A. The DTDs for the *description* class and the *instructions* class are given in Figure 4-4.

In these type definitions, a description is a PCDATA field (which means that it contains raw data) with one of five attributes: intro, summary, conclusion, explanation or preformatted. The first four attributes allow the instructional designer to provide detailed meta-data about the semantics of any particular description instance. The fifth attribute is an artifact that proved expedient for the prototype implementation. The default attribute for a description is explanation. *Instructions* are defined as a list that must contain at least one *description* instance and then a set of one or more *instruction steps*. *Instruction step* may also contain a *description* and must contain a single *action*. An *action* is a text statement. The interesting thing about the document type definitions is that they naturally allow for hierarchical descriptions of data and for reuse of previous

definitions. For instance, many of the instructional classes defined use the *description* element. Because the definition of the data is separate from the display characteristics, simple re-use of the definitions is possible. XSL allows the styling of the data to change depending on the context, but the information stays the same. This feature alone makes XML and XSL appropriate for use within APHID.

4.2.3 Data Elements

The second part of the data model for the hypermedia applications is composed of the meta-data for the data elements. Although data elements are instantiations of instructional classes, they form a large portion of the hypermedia application model from the perspective of the instructional designer. The instructional designer defines

```
<!ELEMENT data_element
        (title?, description?,
                (
                question_list|example|instructions|index|simulation|
                practise_problem|quiz|FAQ|description|narrative
                )
        )>
<!ATTLIST data_element
        id ID #REQUIRED
        name CDATA #IMPLIED
        importance (critical | explain | enrich) "explain"
        level (beginner | intermediate | advanced) "beginner"
>
```

Figure 4-5 Data Type Definition for Elements

and organizes the data elements. A data element can be considered to be a class that inherits from both instructional classes and the concepts that make up the domain of instruction. It inherits structure from the instructional class and meaning from the domain concept. It is possible to create a model of the application using template data elements and add the actual content later in the development process. The document type definition for data elements is shown in Figure 4-5.

A data element may have a title and a description and then must consist of one instance of an instructional class. In addition a data element has a set of meta-data including a

unique ID, a name, an importance rating and a level rating. The name makes it possible to display something meaningful in a list-view of data elements. The importance and level ratings are critical to the hypermedia application construction algorithms.

The *importance* of a data element indicates the priority that should be placed on ensuring that the particular data element is included in a hypermedia application. Elements that are of critical importance contain basic information that is necessary to an understanding of the topic domain. Elements that are explanatory may contain basic information, but are written with more depth than critical elements and may also have some information that is optional. Enrichment elements contain interesting information about the topic that is not considered to be crucial to an understanding of the domain.

The *level* of a data element indicates the type of learner for which the element is most likely to be appropriate. Elements at a novice level will typically be written in simpler language than elements at the intermediate or advanced levels. Novice level elements are usually shorter and contain fewer sentences or ideas than intermediate or advanced elements. Both the importance and the level attribute are used by the software to guide the selection of data elements for the hypermedia applications.

The APHID software is designed to facilitate data element construction using simple edit boxes and drop-down lists for attributes. The actual data elements are stored as XML documents while the meta-data is stored along with the other attributes of the model. Any elements that are not textual in nature (i.e. images) are stored separately and the file name and meta-data is stored as an XML document. Because XML is the storage mechanism, the data element editing portion of the prototype software functions exactly like an XML editor. To avoid re-implementing existing software a third party XML editor, Xeena from IBMAlphaworks (IBM Alphaworks 2000), was used for XML editing rather than an editor embedded in the prototype. Figure 4-6 shows the dialog window that users work with to create new data elements. The left-hand side of the screen lists the data elements that are already created and the dialog for creating new elements is on the right-hand side of the screen.

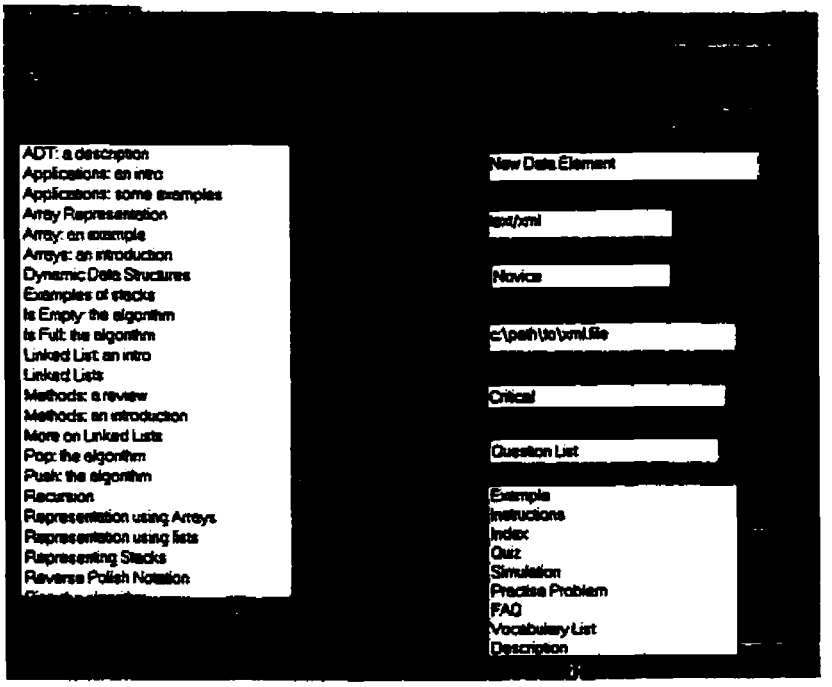


Figure 4-6 Element Creation Dialog

The user can select the level, importance and instructional class from drop-down lists. The Display field becomes the name attribute of the element (and is, in fact, also used as the title in the prototype software). The FileName attribute points to the name of the XML file associated with the data element. In this implementation the XML file is created using a third-party XML editor. A sample XML file for one of the data elements used in the stacks tutorial is given in Figure 4-7.

```
<vocabulary_list> <vocabulary_item>
<term>Stack </term>
<definition> A stack is a homogeneous collection of items of any one type, arranged linearly with
access at one end only, called the top. This means that data can be added or removed from only the
top. Formally this type of stack is called a Last In, First Out (LIFO) stack. Data is added to the stack
using the Push operation, and removed using the Pop operation. </definition> </vocabulary_item>
</vocabulary_list>
```

Figure 4-7 Example XML Source for Data Element

4.2.4 Semantic Model

The APHID development process uses a semantic model to define the domain of instruction independently from the data that presents the concepts to the learner. The separate modelling of the semantics of the domain facilitates the reuse of elements and instructional items within different hypermedia applications. Semantic models are not new to instructional design, but are not frequently incorporated into hypermedia design.

The semantic model used by APHID is represented as a concept map with four types of relationships between entities on the map. The graphical representation for the map is based on a simplified class-diagram representation. The simplified representation shows concepts as classes with no attributes or methods and allows for abstraction, aggregation, and prerequisite relationships between concepts. The representation is not quite accurate from a software design viewpoint, since the concepts do have attributes and even methods after a fashion, but a complete class-diagram would be inappropriate for the instructional designer using the APHID design environment.

4.2.4.1 Concepts

Concepts form the basis for the semantic model. A concept is an idea or unit of knowledge that has cohesion and a sense of belonging together. A dictionary definition for concept is as follows: "*an abstract or generic idea generalized from particular instances*" (Merriam-Webster Online 2000). For the purposes of building concept maps, concepts can be quite broad or general, and often have sub-concepts, but should be able to be described succinctly in a sentence or two.

Four types of concepts have been defined for use within the prototype APHID software. The four types allow for enough distinction between concepts to allow the hypermedia construction algorithms to function adequately without overwhelming the instructional designer with choices. The four types are: root, seminal, secondary, and target. The root concept is the main concept for the domain of instruction. For example, if a concept map were being created about different modes of transportation, the root concept would likely be transportation.

Seminal concepts are those concepts which are key to understanding the domain. If a completed concept map were stripped of all concepts except the root and seminal concepts, there should still be enough information in the map to provide a good, general picture of the domain. Secondary concepts are concepts that are not critical to a general understanding of the domain but that are necessary for a detailed understanding.

The notion of target concept is more related to instruction than to the semantics of the

Figure 4-8 Concept Editing Dialog

domain. A target concept is one that is the focus of the instruction. A target concept will be explored in more detail than other concepts, and often will be returned to frequently during the course of a tutorial. It is possible for the instructional designer to change the type of a concept within the environment. Figure 4-8 shows the dialog box for editing concepts. The user may select from the four types of concepts, change the name of the concept and observe the number of links to the concept and the number of elements associated with the concept. The latter two attributes are intended only as information for the instructional designer as the design of the hypermedia application proceeds; they cannot be changed from within this dialog. Figure 4-9 shows the concept

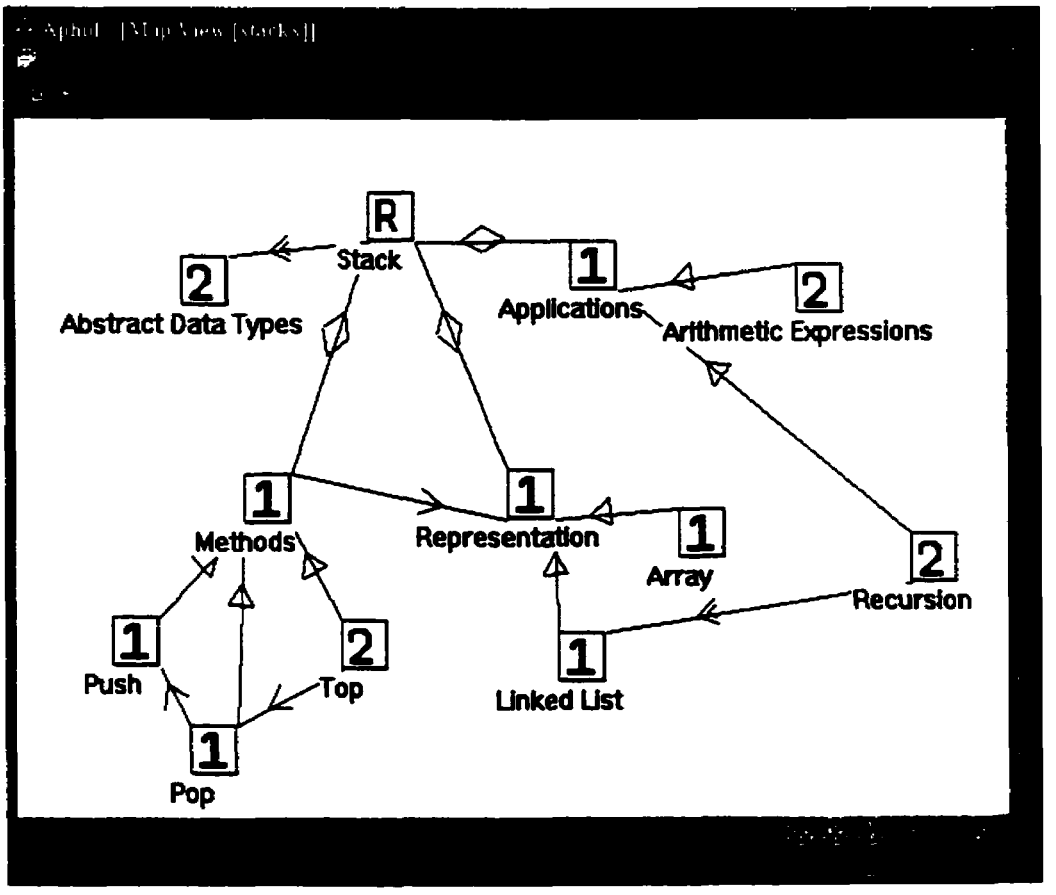


Figure 4-9 Concept Map for Stacks Example

map for the stacks example. In the figure, concepts represented by a 1 are seminal concepts and concepts represented by a 2 are secondary concepts. The root concept is shown by the R icon. There are no target concepts on this concept map

4.2.4.2 Links

Links between the concepts define the relationships between ideas and add structure to the domain definition. The process of identifying relationships between concepts increases the instructional designer's understanding. The relationships are the mechanism for providing guidelines for the construction algorithms so careful identification of relationships is crucial if good hypermedia applications are desired.

There are four types of relationships between concepts, with the roles of each concept within the relationship playing an important part in the design. Facility is provided within the prototype software for the aggregation, abstraction, uses and prerequisite

relationships discussed in Section 3.3.2. Links are created within the prototype environment by dragging a line between two concepts. The completed concept map for the stacks example is shown in Figure 4-9. The symbol on the link between the concepts differentiates the types of links. The symbol is closest to the concept that is on the 'receiving' end of the link; the concept that would be the object in a sentence about the link. For instance, the concept of applications *is part of* the concept of stack- so the link between them points towards stack. The type of a link can be changed by the instructional designer, but the only way to change the direction of a link is to delete it and draw it the other way around.

4.2.4.3 Adding Data Elements

Once the concept map is created and the data elements are defined the final step in creating the semantic model is to associate data elements to concepts. The associations between elements and concepts are not typed and an element can be associated with more than one concept. Multiple associations are likely in the case where an element is a simulation or demonstration that illustrates many concepts in the domain. One such element from the stacks example is the applet that demonstrates all of the stack operations.

Elements are dragged from the element list to a tree representation of the domain model to form the associations. The interface can be seen in Figure 4-10. The list of elements is on the left and the tree representation of the concepts and elements is on the right. From the tree representation, the designer can check on the properties for both concepts and elements. Once the instructional designer is satisfied with the number of elements associated with each concept, hypermedia applications can be generated.

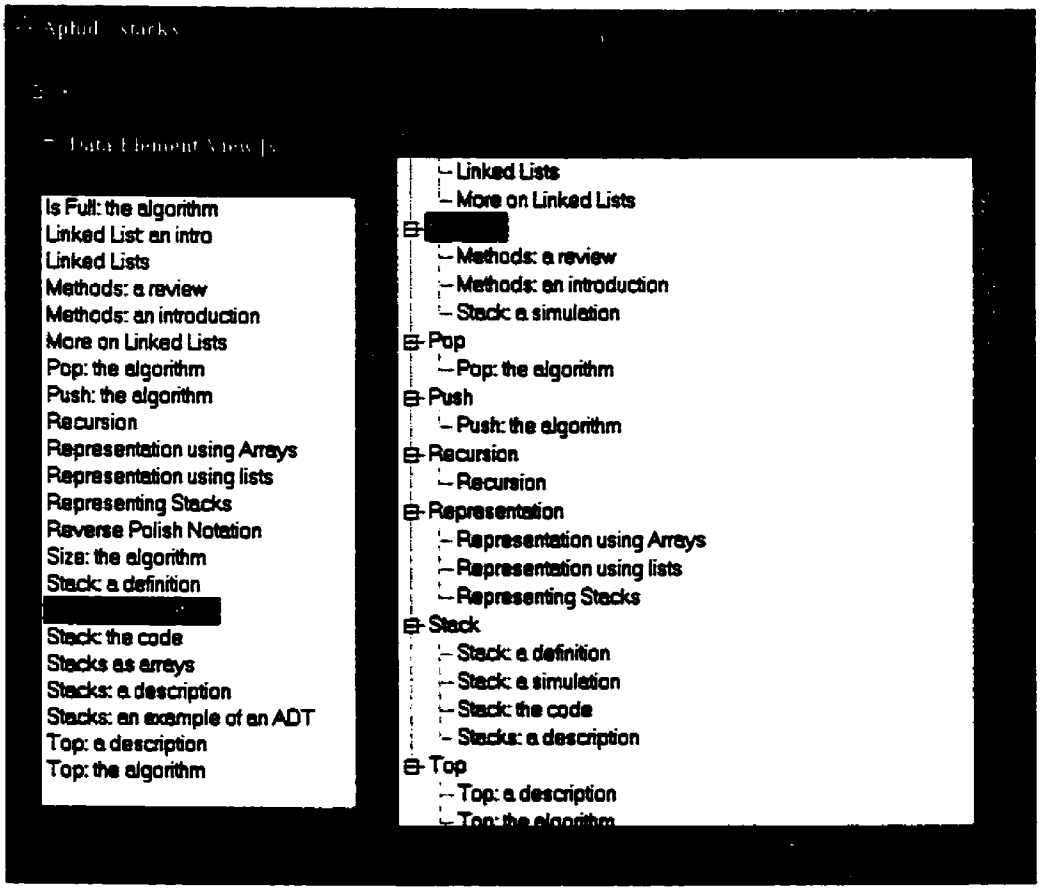


Figure 4-10 Associating Data Elements and Concepts

4.3 Creating Hypermedia

From the instructional designer's point of view, the creation of a hypermedia application from the completed model can be as simple as setting two parameters (type of application and level of user) and clicking a button. The APHID environment decides which concepts to include, which data elements to include and which hyperlinks to create between pages based on the model and the supplied parameters. The actual processing that occurs is much less simple, however. This section discusses how the algorithms behind the creation process work and how the different parts of the model are chosen or excluded from the finished application.

The basic process is quite straightforward. Given a graph-representation of a domain, traverse the graph and at every node of the graph determine if the concept represented

by that node should be included or excluded from the final application. If the concept is included, select the appropriate data elements for the concept, select the necessary hyperlinks to other concepts, build the HTML page and continue. When the traversal is finished, the web site will be constructed. Complexity is added when constraints to ensure quality in the web site are added.

These constraints have default values selected when the instructional designer chooses an instructional pattern and type of user. The constraints control things like the number of elements on a page, the number of pages allowed and the kinds of data-elements selected. APHID is designed to allow the instructional designer to select values for some or all of these constraints manually. This feature allows experienced designers to create hypermedia applications that accommodate their personal preferences. For instance, the default values for the constraints represent the personal preferences of the author. They are based on accumulated experience working with the subject matter and learners as well as upon personal teaching style. Other educators will have different preferences. A more complete discussion of the constraints involved can be found in Section 4.3.1.2.

The prototype software divides the work of creating the site into two discrete steps. The first step is to create a navigational model and to record that model. The second step is to transform the navigational model into HTML. The second step is accomplished by applying an XSL style sheet to the XML representation of the navigational model. The first happens within the modelling environment when the instructional designer selects the 'build web site ' option. The two steps were separated because third-party processors for XML and XSL were freely available, but were written in Java while the rest of the prototype software is written in C++. Although integration of software written in the two languages is possible, a simpler solution is to create the hypermedia application using a two-step manual process.

4.3.1 Building the Navigational Model

The APHID prototype software represents the navigational model internally rather than explicitly presenting it to the instructional designer. The navigational model records all

of the entities that will ultimately be seen in the final hypermedia application including concepts, hyperlinks, and data elements. The information about the navigational model is stored in a separate data structure created during the execution of the build algorithm. The data structure is a list of *navigational nodes* stored in tutorial-order, the order in which they would be presented to the learner if a strict next-back tutorial were being created. The creation of each node is subject to a number of constraints and algorithms. This section first presents relevant details about the navigational nodes and the information kept about each node. Then the variables that govern the selection of items for the navigational model are discussed, along with a description of their effect on the final hypermedia application. The section concludes with presentations of the algorithms used to select entities for the model.

4.3.1.1 Navigational Nodes

Each navigational node in the model represents a single hypermedia page in the finished application. Any concept found in the domain model will be represented by one or more navigational nodes if the concept is selected to be in the final application. The C++ header information for the navigational node class is presented in Figure 4-11. It illustrates the information that makes up the navigational node class. The class keeps track of the concept it represents, the links and data elements that are associated with it and some statistical data such as the depth from the root node, and the number of other prior nodes that the same concept has representing it.

```
        class CNavNode : public CObject
        {
//attributes
        public:
                CConcept * m_pcConcept;  //concept in this node
                CObArray  m_aLinks; //links that should be associated with this node
                CObArray  m_aElements; //data elements used for this node
                int m_iDepth;  //depth from root
                CRules* m_pRuleSet;  //set of rules associated with the navigational model
                CString m_sTitle; //title for the node, and ultimately the hypermedia page
                int m_iExpansionNum; //this node is the nth presentation of its concept
        private:
                CObArray m_aFromLinks;  //points to CNavNodes that point at this node
                CObArray m_aCriticalElements;  //keep the data elements sorted until write time
                CObArray m_aExplainElements;
                CObArray m_aEnrichElements;
//methods intentionally not presented here
        };
```

Figure 4-11 C++ Header for CNavNode

The notion of prior nodes is an important one. Any concept in the domain model can be represented by a number of nodes. The final navigational model is a list of navigational nodes in *tutorial-order*, the order in which nodes would be presented if a linear tutorial were constructed. A *prior node* in this context is an earlier node (in the tutorial-order) that is about the same concept. A simple explanation is that the prior-node represents part 1 of a concept explanation and the subsequent nodes represents part 2 (if there are only two parts). Keeping track of this information makes it possible for the prototype software to customize presentations of the concept to reflect their position in the tutorial. For instance, nodes that appear later in the tutorial might have hyperlinks leading to earlier explanations. Or later nodes might have the more difficult elements while earlier nodes present the easier elements. Much of the tailorability of the final hypermedia applications comes from the software's ability to build on this type of information.

4.3.1.2 Rules for Navigational Model Construction

Typically the navigational model is constructed using preset values for the different rules associated with application construction. These values are determined by the two attributes of the application, *User* and *Instructional Strategy* or by manual intervention

on the part of the instructional designer. Manual intervention presently requires minor modification to the software. A user interface to accomplish this is trivial to implement but was not important for this research.

The *User* attribute is chosen to be one of novice, intermediate or advanced. The level of the user determines which data elements are selected and the number of pages and elements that a web site will have. It also partially determines which concepts will be selected to be included in the data model. The *Instructional Strategy* attribute determines the tutorial-order for the navigational model as well as the hyperlinks and the order in which they appear on the web pages.

Any number of instructional strategies are possible; four have been implemented in the prototype software: Depth First Instruction, Spiral Curriculum, Remedial Instruction and Review Instruction. The four strategies are implementations of the organizational patterns discussed in Sections 3.4.2. Depth First Instruction and the Spiral Curriculum were discussed in Section 3.4.2. The Remedial Instruction strategy is intended to provide remedial teaching for a particular sub-concept along with a quick review of the other main concepts. The Review Instruction strategy is intended to provide a high-level review for an entire topic.

Once the values for the rules are determined, the set of rules created for the navigational model determines the ultimate size and structure of the application. The C++ header file for the rules class is shown in Figure 4-12. The rules class records the user type and translates the instructional strategy attribute set by the instructional designer into the appropriate organizational pattern. The rules are divided into several sets: rules about site size, rules about page size, rules about concept selection and rules about data element selection. The rules class could also record rules about selection of links. The prototype contains the link-selection logic in one section of the code, rather than in the rules class, because a non-optimal choice for internal storage of links was made early in the prototype design. It proved to be simpler to implement link selection directly, rather than make awkward modifications to the rules class.

```
class CRules : public Cobject
{
        private:
                UserType m_eUser;
                OrgPatternType m_ePattern;
/***     site limitations    ***/
                int m_iMaxDepth;
                int m_iMinDepth;
                int m_iMaxPages;
/***     page limitations    ***/
                int m_iMaxLinks;
                int m_iMinLinks;
                int m_iMaxPics;
                int m_iMaxText;
                int m_iMinText;
/***     rules about concepts        ***/
                int m_iRootMaxExp;
                int m_iSeminalMaxExp;
                int m_iSecondaryMaxExp;
                int m_iTargetMaxExp;
                TraversalType m_eRootTraversal;
                TraversalType m_eSeminalTraversal;
                TraversalType m_eSecondaryTraversal;
                TraversalType m_eTargetTraversal;
                AlgorithmType m_eRootAlgorithm;
                AlgorithmType m_eSeminalAlgorithm;
                AlgorithmType m_eSecondaryAlgorithm;
                AlgorithmType m_eTargetAlgorithm;
                bool m_bRootMarked;
                bool m_bSeminalMarked;
                bool m_bSecondaryMarked;
                bool m_bTargetMarked;
/***     rules for data elements     ***/
                int m_iMaxElements;  //should be normalized
}
```

Figure 4-12 C++ Header for Rules Class

The rules about size, both page size and site size, are mainly for use by constraint checking algorithms, although they are used somewhat during the building of the navigational model. The algorithms to construct the navigational model use the rules about concept and element selection as well as the implicit rules about link selection. The choice of user-type affects the rules for size, for instance, a site designed for novice users would be restricted in size more than a site designed for advanced users.

The choice of organizational pattern affects rules that determine the selection of items for the final web-site (user-level has some effect on these rules as well). The rules about concept selection include information about how the domain model should be ordered for that particular type of organizational pattern (algorithm type and traversal type).

This information is used by the APHID software to determine how the domain model (made up of concepts, links and associations to elements) will be processed to create the navigational model.

An example is instructive. Consider an application intended to present the material to a novice learner who has previously studied the concept. This learner has demonstrated difficulty with the concept of *methods* and needs remedial instruction in that concept as well as a review of other concepts. Such an application would use the remedial instruction pattern, which suggests that all concepts should be marked for expansion and that all but the target concept use the same traversal algorithm. The target concept (methods in this case) uses a different traversal algorithm to present the target material in smaller sections. The values for the rules are determined and the rules are methodically applied to each entity in the domain model. The result is an annotated domain model that supplies information about how those entities should be treated when

Table 4-1 Concept Based Rules for Remedial Application

Rule	Root Concept	Seminal Concept	Secondary Concept	Target Concept
Maximum Expansions	1	1	1	3
Traversal Type	Prefix	Prefix	Prefix	Prefix
Algorithm Type	Depth	Depth	Depth	Breadth
Concept Type Selected	Yes	Yes	Yes	Yes

the navigational model is constructed. The values given the four different types of concepts for this example are shown in Table 4-1. For all of the implemented strategies, the root concept and seminal concepts are handled identically. The distinction between them is necessary for the graph traversal algorithms, and it is conceivable that additional instructional strategies could be developed which treat the two differently.

In addition to values that depend on the type of concept involved, some rules are global to the intended hypermedia application. Generally these global rules are based more on

Table 4-2 Example Global Rules for Remedial Application

Rule	Value
Maximum Links	4
Maximum Elements	4
Maximum Depth (of tree)	5

the type of user the application is intended for than on the instructional strategy chosen for the application. The values set for a sampling of these rules for the example application are given in Table 4-2. It should be noted that many additional rules are possible, including rules to set the minimum numbers for links and elements, the total number of pages allowed in the finished application and rules describing the desired balance between text and other media. The set used in the prototype software is rich enough to allow reasonable control over the application creation process without creating a rule set that has conflicts requiring complex constraint solving algorithms. Further development of the software will require a reassessment of the rules involved and possibly the introduction of new rules in addition to the simple ones presently used.

4.3.1.3 The Traversal Algorithms

Once all of the concepts have been annotated with values for the rules, the traversal is begun using the traversal types given to the concepts during the marking phase. The root concept of a graph must have a traversal type, which then becomes the default traversal for the application. If a sub-concept has a different traversal-type than its parent, the entire sub-tree rooted at that concept will be processed using the sub-concept's traversal algorithm. It is possible that concepts will appear several times in the list if their maximum expansion attribute is set to any number greater than one. Concepts that are not marked during the first phase of the application generation process are ignored during this phase. The order created determines the order in which concepts will be expanded and also determines, in part, which concepts will have hyperlinks to other concepts. The prototype software is limited to either a Depth First or a Breadth First traversal of the graph, although there is no reason why more sophisticated

algorithms couldn't be experimented with. The software is designed for the addition of alternate traversal algorithms.

In addition to varying the algorithm for traversing the graph, the order of traversal can be varied. The software is designed to handle infix, prefix, postfix and a combination prefix-postfix order. A typical classroom teaching scenario generally involves a prefix expansion of concepts. The teacher usually explains the current concept before moving on to child concepts. It is possible to imagine review situations where concepts are presented in a sort of post-fix order, or in a combination of prefix and postfix where the general concept is discussed, the child concepts are discussed and the general concept is reviewed. The prototype implementation of APHID uses only prefix traversal order because it was adequate for demonstrating the principles.

The first step in creating the navigational model is to do a full traversal of the graph represented by the domain model. The graph is rooted by the root concept and the algorithm initially used for the traversal is determined by its algorithm-type value. Recall the stacks example. Root, seminal and secondary concept types were set to use a Depth First Search and target concepts were set to use a Breadth First Search to allow multiple expansions of the concept (which breaks up the information into smaller chunks and presents more detail than a single page. Each time a concept is encountered during the traversal it is added (in prefix order) to the list of concepts. At the end of the traversal, the list of concepts represents the *tutorial-order* for the finished hypermedia application, or the order in which the pages would be presented if a strict tutorial were being generated. The tutorial-order list of concepts that makes up the first stage of the navigational model for this application is given in Figure 4-13.

1.Stack	2.Applications
3. Arithmetic Expressions	4. Recursion
5.Representation	6. Linked List
7.Array	8. Methods
7.Push	10.Pop
11.Top	12. Methods
13. Push	14. Pop
15. Top	

Figure 4-13 Tutorial Order List of Concepts for Stacks Example

To streamline the traversal algorithms, rules about the format of the graph must be enforced. The graph may have loops, but there may not be loops caused by prerequisite link types. The graph must have a single root element. It is not a requirement that all nodes on the graph be connected, in fact if a concept is represented on the graph without any links to or from other concepts it will still be represented in the ordered list of concepts. Its placement in that list will be arbitrary however, since it has no relations to help determine the semantics of the concept in relation to the other concepts.

Of some interest is the ability of the software to shift from one algorithm to another in the midst of a graph traversal. If the instructional strategy chosen set the algorithm for one type of concept to depth and the algorithm for a second type of concept to breadth, a mid-traversal shift would occur. This is the case with the remedial instructional strategy.

The run-time changing of algorithms was accomplished using inheritance and static and virtual methods within the parent class. The parent class is called CGraphWalker with two key attributes: an array containing the partially ordered set of concepts, and the type of traversal for that particular graph-walker. GraphWalker has two methods of interest: a virtual method, walk, which represents the algorithm for traversing, and a static method, recurse, which handles the switch between the algorithm types. Presently two classes inherit from GraphWalker: CBreadthWalker and CDepthWalker. Each implements the appropriate algorithm in their walk method. The CDepthWalker class uses recursion to handle the list of child nodes requiring processing. CBreadthWalker has an additional static queue that stores that information. The method for using these classes is to define a GraphWalker and call it with the root concept. The recurse method is called for children of the root concept and processing proceeds. The specific algorithm used for the recurse method is given in Figure 4-14. Note that the method is called with a parameter pointing to the child concept that will be recursed on. The algorithms for the walk methods are reasonably straightforward implementations of Depth First and Breadth First traversals.

```
if (child.concept sent as a parameter is to be included in the application)
{
        switch (child.concept_Algorithm_Type)
        {
        case DEPTH:
                {
                        newWalker = new (DepthWalker);
                        newWalker.walk(child.concept);
                        ordered_array.append (newWalker.ordered_array);
                        break;
                }
        case BREADTH:
                {
                        newWalker= new (BreadthWalker);
                //make sure the concept is on the static queue for the breadthwalker
                        CBreadthWalker::Queue.InsertAt(0, child.concept);
                        newWalker.walk(child.concept);
                        ordered_array.append (newWalker.ordered_array);
                        break;
                }
        }
}
```

Figure 4-14 CGraphWalker Recurse Algorithm

When a shift of algorithm is detected, the node with the new algorithm and all of its child nodes are processed with the new algorithm type. Child nodes are nodes in the sub-ordinate end of an abstraction or aggregation relationship. Returning to the example, the target concept is set to the concept of Methods. The rules for this application state that the target concept uses the breadth-first traversal algorithm. So, when the Methods node is reached, it and all of its children (Push, Pop, Top) are processed using the breadth-first algorithm. Once the nodes for the final application are created and the tutorial order for the concepts is calculated, the process of choosing links and elements for each navigational node is begun.

4.3.1.4 Selection Algorithms

Navigational nodes contain concepts, links and data elements. After the traversal phase, the navigational model contains a list of nodes and associated concepts but no data elements or links. The algorithms for link and data element selection use the rules associated with the hypermedia application construction to guide the selection. The algorithms used in the prototype software are simple, yet the hypermedia application

that results is still acceptable. Complex algorithms are conceivable and would likely result in highly tuned hypermedia application.

Recall that the domain model created by the instructional designer can contain four different types of associations (aggregation, abstraction, prerequisite and uses) with two roles for each type of association for each concept. APHID defines two additional types of associations between navigational nodes, again with two roles for each type, bringing the total possible number of roles to 12. Each role represents a different category of hyperlink in the finished hypermedia application. Any single node in the navigational

Table 4-3 Roles and Association Types

Association Type	Role Label
Aggregation	Part
Aggregation	Whole
Abstraction	Example
Abstraction	General
PreRequisite	Presib
PreRequisite	Postsib
Uses	Successor
Uses	Predecessor
OtherExpansion	Parent
OtherExpansion	Child
TutorialOrder	Next
TutorialOrder	Back

model may participate in any of these association types in either role. The roles also determine which hyperlinks will appear in the final application. Table 4-3 lists the possible roles for each type of association.

In the stacks example, the navigational node that represents the concept of *Representation* participates in a number of different associations (refer to Figure 4-9). It is in an aggregation relation with *Stacks* and has the role of **Part**. It is in an

abstraction relation with both *Linked List* and *Array*, having the role of **General** in both associations. It is also in a prerequisite relation with its sibling concept *Methods* and has the role of **PreSib** (the pre-requisite sibling) in that relation. The same navigational node is also in a tutorial-order association with two concepts, *Top* and *Linked List*. In the association with *Top* it has a **Back** role and in the association with *Linked List* it plays the **Next** role. Since the instructional strategy for this example is Remedial and this node is not representing the target concept, there are no other nodes representing the same concept so the node does not participate in an other-expansion association. In total, this one navigational node participates in six associations with five different roles.

APHID uses this information in conjunction with the user type and organizational pattern information to select associations to appear as hyperlinks on each page of the hypermedia application. The total number of links on any page is bounded by the maxLinks value in the rules for the application. The selection algorithm presently chooses some of the roles based on the user-type and some based on the instructional strategy. Table 4-4 shows how the default application rules within APHID select

Table 4-4 Association Selection Rules

	Based on Instructional Strategy					Based on User	
Role Label	DFI	Spiral	Remedial Y=target	Review	Novice	Intermediate	Advanced
Part	X	X	X Y	X			
Example	X	X	X Y	X			
Whole	X		X Y				
General	X		X Y				
Next	X	X	X Y	X			
Back	X	X	X Y	X			
Successor			Y				
Predecessor			Y		X	X	X
Presib					X	X	X
Postsib						X	X
Parent					X	X	X
Child							X

associations to be included in the hypermedia application. After the list of possible hyperlinks is generated, the maxLinks rule is applied and potential links are removed if

necessary. **Next** and **Back** roles are always included, but other types may be removed if too many link possibilities exist for a particular page.

Data Elements must also be selected for each navigational node. Data Elements are selected in a similar fashion to links with two key differences. First, unlike hyperlinks, a data element cannot appear more than once in the hypermedia application so must be removed from the list of available elements once it has been selected for inclusion on a hypermedia page. Most often a data element is associated with only one concept and this is not an issue, but some data elements (such as simulations that span a number of concepts) may be associated with multiple concepts. In this instance the first navigational node that includes the element marks it as used, and other nodes simply do not use that element. Second, when several navigational nodes represent the same concept, such as in a spiral presentation of the material, data elements must be divided more or less evenly between the successive presentation of the concept. Also, the elements should be presented so that a tutorial-order viewing of the different nodes for the concept presents the material in increasing difficulty and complexity.

The prototype software accommodates these needs by keeping an ordered list of possible elements for each concept. When an element is selected for use with a navigational node, it is taken out of the ordered list for that concept. When a spiral presentation of a concept is being prepared, each navigational node in the set for the concept is allowed to select only a fraction of the elements left in the concept's list (based on how many pages are yet to be created for that concept).

Data elements have two attributes each with three possible values. An element can be at the novice, intermediate or advanced levels and can either be critical, explanatory or enrichment material. Data elements are also classified by their instructional class (e.g. quiz, simulation, description) and the intention was to also use the instructional class information to organize the selection of data elements for a single concept. In reality, there weren't enough data elements associated with a single concept, nor a wide enough variety of types of data elements in the applications developed for this research to test that idea. There are nine classes of data elements and simply identifying the elements as belonging to a particular class proved to be enough distinction. The notion of

organizing data elements with relationships and rules is still interesting and might prove useful in future implementations.

Table 4-5 shows the data elements that would be selected (and the order in which they are chosen) based on the attributes of the data elements. In the table the level of the element is denoted by N, I or A for novice, intermediate or advanced. The importance of the element is denoted by C, X or E representing critical, explanatory or enrichment. For instance, the notation NE would indicate an enrichment element at the novice level. When building hypermedia applications using the built-in application types, APHID selects elements using both the user type and the instructional strategy as a guide. All of the elements of one type are chosen before moving on to element of the next type.

Table 4-5 Data Element Selection

	Novice	Intermediate	Advanced
DFI	NC, IC, NX, IX	NC, IC, NX, IX ,NE ,IE	NC, IC, AC, NX, IX , AX, NE ,IE, AE
Spiral	NC, IC, NX, IX	NC, IC, NX, IX ,NE ,IE	AC, IC, NC, AX, IX, NX, AE, IE, NE
Remedial	NC, IC, NX, IX	AC, IC, NC, AX, IX, NX	AC, IC, NC, AX, IX, NX, AE, IE, NE
Review	NC, IC	AC, IC, NC, AX, IX, NX	AE, IE, NE, AC, IC, NC

4.3.1.5 Validation and Constraint Enforcement

The prototype does not yet validate the initial data model, which means that the instructional designer must ensure that the graph representation of the domain adheres to the rules. Validation of that graph is an unimplemented feature left for future versions. APHID does restrict the addition of concepts, links and data elements during the navigational model creation phase to ensure that the rules are adhered to. Even so, after the model is completed, it is re-checked to ensure that no rules have been violated. As more complex constraints are added to the software, this validation phase will become

more important and more complicated. The prototype software automatically ensures that size restrictions are not violated and makes sure that all links point to valid nodes. For instance, a link could point to an invalid node if the node it had been pointing to were deemed to have too few data elements and was removed from the model.

After all constraints and rules are satisfied, the prototype software creates an XML document from the navigational model. Figure 4-15 shows the XML document for one node of the stacks example. The complete listing is given Appendix B. The example in the figure illustrates how the links are typed, how url and link-text are incorporated and how data elements are included in a page. No meta-information is added to the data element tag, simply because none was used by the prototype software. The construction of this document represents the completion of the modelling portion of the APHID hypermedia development process. The next stage is to create a presentation of the model for users.

```
<page ref="Stack" title="Stack"><links>
<link type="NEXT"><url>Applications.html</url><link_text>Applications</link_text></link>
<link type="PREDECESSOR"><url>AbstractDataTypes.html</url><link_text>Abstract Data
Types</link_text></link></links>
 <element_list>
<data_element><title>Stacks: a description</title> <example><instance type="positive" media="text">
In order to clarify the idea of a stack let's look at a "real life" example of a stack. Think of a stack of
plates in a high school cafeteria. When the plates are being stacked, they are added one on top of each
other. It doesn't make much sense to put each plate on the bottom of the pile, as that would be far more
work, and would accomplish nothing over stacking them on top of each other. Similarily when a plate is
taken, it is usually taken from the top of the stack.</instance><instance type="positive"
media="image">plates.gif</instance></example></data_element> <hr> </hr></element_list> </page>
 <page ref="Applications" title="Applications"><links>
<link type="NEXT"><url>Representation.html</url><link_text>Representation</link_text></link>
<link type="BACK"><url>Stack.html</url><link_text>Stack</link_text></link>
<link type="WHOLE"><url>Stack2.html</url><link_text>Stack: Part 2</link_text></link></links>
 <element_list>
<data_element><title>Examples of stacks</title> <list><description>Examples of stacks </description>
<list_item>a pile of papers </list_item>
<list_item>Cars in a non-circular driveway</list_item>
<list_item>Nested function calls </list_item>
<list_item>Code formatting and expression syntax</list_item>
<list_item>Evaluation of arithmetic expressions </list_item>
</list>
</data_element> <hr> </hr>
<data_element><title>Applications: an intro</title> <description type="intro">Stacks are used in many
applications, including function calls and recursive calls. Stacks are also used when processing
expressions such as arithmetic expressions in polish or reverse polish notation.</description>
</data_element> <hr> </hr></element_list> </page>
```

Figure 4-15 Example XML document

4.3.2 Building the Presentation

The APHID process uses instructional patterns (recorded as instructional strategies) to guide the structural development of the hypermedia application. It uses presentation patterns to guide the presentation of the finished application to the user. Ideally the software would use the same sort of selection process for choosing the method of presentation as it does for selecting links and data elements, by selecting patterns based upon the level of the user and the type of application desired. However, that sort of approach requires an in-depth understanding of user interface design. It is beyond the scope of this thesis to delve into user interface design in detail.

Yet, it is impossible to determine if hypermedia applications are useable and well-constructed without a user-presentation through which to view the application. The compromise for this thesis has been to create one set of presentation patterns that are

used for all hypermedia applications, regardless of user type or instructional strategy. The patterns are encoded using the eXtensible Style Language (XSL) which allows for simple extension of existing patterns and addition of new ones..

The remainder of this section will present a brief introduction to XSL and then discuss how XML and XSL are used to encode the presentation patterns used to create hypermedia applications with APHID.

4.3.2.1 XSL

The eXtensible Style Language(XSL) consists of two separate parts: a formatting vocabulary which is as yet incomplete, and a transformation language (XSLT) which is used to specify how one could mechanically change one XML document into a different XML document (XSL Transformations (XSLT) 1.0 1999). Such transformations have many uses, such as when only a portion of the original document is needed, when the information is needed in a different order, or when information from multiple documents must be merged into a single document (possibly with completely different XML tags).

XSLT can be used to describe which parts of an XML document should be placed in a new XML document and to describe the order in which they should be put in the new document. It has facilities for sorting data as well as for conditionally selecting data.

XSL is a collection of template rules. Each rule consists of a pattern and a template. The pattern determines which XML elements from the input document will be processed using the template. An XSLT document usually consists of at least one template for every type of element in the XML document. Every input element is processed with a single template. Each pattern takes an element from the input, applies the template, recursively processes any children, and creates XML output by combining the output from the children and the current node (Harold 1999). The information represented in the XML document is processed in depth-first order.

The APHID prototype uses XSLT documents to define the presentation for individual elements within the application and a Cascading Style Sheet (CSS) to control font, color and background. The use of the style sheet in addition to XSLT allows the software to

specify formatting as well as selection and presentation even though the formatting specification for XSL is still undefined.

4.3.2.2 Presentation Model

The XSLT portion of the presentation model consists of a set of templates or patterns for defining the presentation of the hypermedia application. The model consists of a pattern for combining elements as well as patterns for each type of instructional class. Presentation patterns must assist the pedagogical motivation for the application while maintaining a good user-interface design for the hypermedia application as a whole.

Each instructional class could be presented in a variety of ways. For instance, a set of step by step instructions can be presented as a list, or as a guided tour. Either the list or the tour is appropriate depending on the length of the instructions, the intention of the educator and whether each step contains a practice element or not. Ultimately the presentation model should depend on the attributes of the application, but the prototype implementation has one presentation model for all applications.

The patterns used in the prototype were defined with the intention of simplifying the interface so that both humans and computers could create hypermedia using the patterns. For instance, at the inception of this research, the in-line placement of hyperlinks was difficult for machine-created HTML, so the presentation model used places all hyperlinks in a separate list. The presentation styles were intentionally kept simple to make the comparison fair for the evaluation study discussed in Chapter 5. Figure 4-16 gives the XSLT template used by the prototype software to create a single page of a web site. Any tags that do not begin with <xsl: are simply copied into the output by the XSL processor. The template first sets up the HTML header and body tags then selects the title attribute of the page from the XML document and outputs that. It then sets up a table consisting of one row and two columns, the leftmost column contains the links for the page and the right-hand column contains the data elements. Control is passed to the template for links and all of the links are processed, then control is passed to the template for the element list and all of the elements in the list are processed. Finally the

```
<xsl:template match="page">
  <html>
   <head>
        <title><xsl:value-of select="@title"/></title>
        <LINK rel="stylesheet" type="text/css" href="../geostyle.css" title="style1"/>
   </head>
   <body>
   <A NAME="top"></A>
    <center>
    <h1>
    <xsl:value-of select="@title"/>
    </h1></center>
   <table cellpadding="5">
    <tr>
          <td VALIGN="Top" >
         <xsl:apply-templates select="links"/>
          </td>
          <td>
         <xsl:apply-templates select="element_list"/>
         </td>
     </tr>
     </table>
        <CENTER><A HREF="#top">top</A></CENTER>
   </body>
  </html>
 </xsl:template>
```

Figure 4-16 XSLT Template for HTML Page

closing HTML tags are added and processing stops.

The patterns for the different types of elements are equally simple. The templates for the element list, the data_element and one element type (the list) are given in Figure 4-17. From the page template, the element list template is called and subsequently calls the data_element template for each data_element in the list. Note that if the element list contained items other than data_elements, those items would not be added to the output when using this template because only the data_element items are processed. The template for data_element, on the other hand, processes every item that makes up the data element (which can be several different combinations- see Appendix A). The difference between the two templates is in the 'select' part of the <xsl:apply-templates> directive.

Finally, the template for list processing selects the description for the list and outputs it, then passes the processing on to the template that handles individual items in the list. Notice that within the templates is the HTML markup (such as paragraph marks <p>) required to format the output as HTML.

Each element defined by the DTD given in Appendix A has at least one style sheet

```
<xsl:template match="element_list">
        <xsl:apply-templates select="data_element"/>
</xsl:template>
<xsl:template match="data_element">
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="list">
      <xsl:for-each select="description">
          <p>
           <xsl:value-of select="."/>
          </p>
       </xsl:for-each>
       <ul>
         <xsl:for-each select="list_item">
  <li>
            <xsl:value-of select="."/>
           </li>
         </xsl:for-each>
       </ul>
</xsl:template>
```

Figure 4-17 Example XSLT Templates

associated with it. The complete listing of the XSL sheet used to create the HTML documents with APHID is shown in Appendix C.

4.4 Example Application

One version of a tutorial about teaching the computer science concept of stacks has been used as an example throughout this chapter. This section will illustrate some of the differences between the different types of applications that APHID can generate by showing parts of some of the other types of web sites possible when using the same model and the same data.

Imagine three different types of applications, the review, the spiral and the in-depth instruction. Further suppose that a novice learner is in need of a spiral introduction to the material, an intermediate learner needs an in-depth treatment and an advanced learner requires a review. APHID creates three quite different hypermedia applications for these three learning situations, for instance, the spiral application has 27 pages, the in-depth application has 16 and the review has 12. Within each page different elements and links are selected for the different types of applications.

The following three figures (Figure 4-19, Figure 4-18 and Figure 4-20) show the format of the first *representation* page in each of the tutorials. The spiral page presents only one data element and links to concepts that are siblings in the graph representation of the domain. The in-depth page presents several elements of data and links to concepts that provide more detail. There are also more links provided because of the more advanced level of the intended learner. The review page presents only a few critical data elements, but chooses the ones that are the most advanced. It also has more links on it to allow the advanced learner more autonomy in choosing a path through the material. Each tutorial presents the links down the left-hand side and has next/back links at the top of the list. The differences between the sites become more obvious when a richer set of data elements is available.

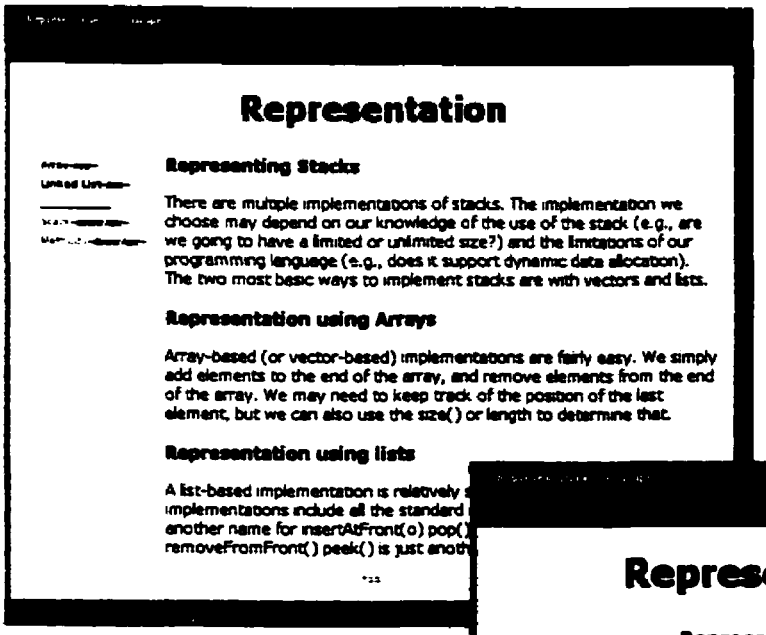


Figure 4-18 Intermediate-Depth

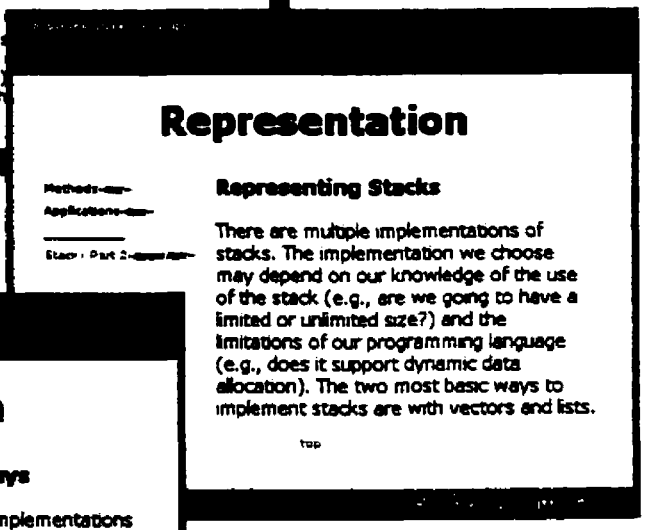


Figure 4-19 Novice-Spiral

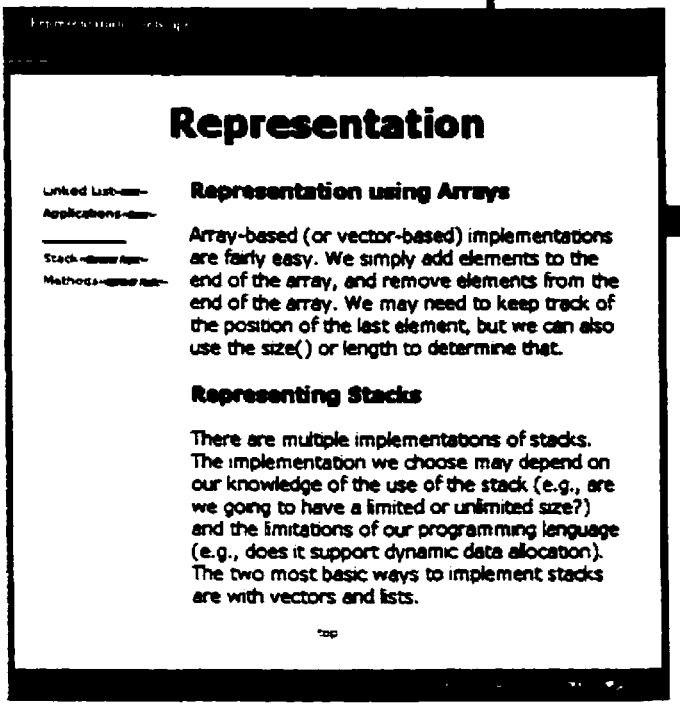


Figure 4-20 Advanced Review

4.5 Summary

The prototype APHID software provides a modelling environment for instructional designers. The process used to create hypermedia applications with APHID mimics the software development process with additional support for the creation of instructional materials. Instructors create an explicit, detailed, model of the domain of instruction that includes a description of the relationships between concepts in the domain. Instructors also identify the instructional materials to be used in the hypermedia application and add those materials to the model of the domain.

Support for instructional design is provided by allowing instructors to vary attributes of the finished web sites according to the specific needs of the students. The different attributes contribute greatly to the final appearance and functionality of the site. Novice designers of instructional material are supported by the definition (and implementation) of four instructional patterns which are presented to the designer as instructional strategies.

The prototype software provides two methods for creating an instructional sequence (depth first and breadth first) and has facilities for easily incorporating more methods as they become developed. The two implemented methods of ordering represent instructional design practices that are commonly used by practicing teachers.

APHID provides a support environment in which instructors can design and create well-built hypermedia applications with a minimum expenditure of time. The resulting applications can be tailored to the specific needs of individual learners, which increases the overall effectiveness of the application.

Chapter Five

Evaluating the APHID approach

A study was performed to evaluate the effectiveness of the APHID approach for designing and producing instructional hypermedia and to determine if it does efficiently produce satisfactory hypermedia applications. During the study, educators who specialized in instructional technology created web sites to be independently evaluated against a set of criteria along with sites created by the author using APHID. The instructional designers kept careful logs of time spent in order to determine the efficiency of APHID versus hand creation of sites. Independently, information about the size and composition of the hypermedia applications was collected to allow comparison between APHID sites and the sites created by the educators. The same information was also collected from a number of existing web sites for further comparison.

This collection of information is used to demonstrate that the sites created using APHID are as good as sites created by hand, are similar in size and composition to hand-created sites, and that using APHID results in a more efficient expenditure of instructional designer time. This chapter presents the details of the study as well as the results of the evaluation.

5.1 Evaluation Methodology

The objective for this research, and the objective of the evaluation was: *to demonstrate that the APHID approach to designing and producing instructional hypermedia can produce satisfactory hypermedia applications more efficiently than humans are able to.*

In order to evaluate whether or not this objective had been met, a study was devised to allow the comparison of instructional web sites created with APHID and those created

without using APHID. All of the sites in the study were created using the same written and visual materials and the same layout for presentation.

The participants in the study were four graduate students or research associates in educational technology, a high-school science teacher who makes use of the web for instructional purposes, a research assistant/analyst with a computer science background, and this author. All participants were familiar with instructional design and with the web.

The four educational technology experts created web sites containing materials collected by the computer science research assistant. They were allowed to use any software they liked, including applications like Page Mill™ (Adobe Systems Incorporated 2000) to create their web sites. Meanwhile, the author created web sites from the same data using APHID. The technology experts were paid for their participation and the results of their work (their final web sites) were not seen by the author until the sites created with APHID were completed.

An expert evaluator was selected because of his expertise in the content area, in instructional design and in using the web. He was not previously known to the author. Although there are risks associated with using a single evaluator, such as the possibility that the evaluator has significant biases towards particular types of hypermedia, the decision was made to go with a single expert evaluator because it proved to be extremely difficult to find willing evaluators with suitable experience and ability.

To evaluate the effectiveness of APHID in creating instructional hypermedia, the evaluator was asked to rate all the sites created without being given any indication that some of the sites had been created automatically. Sites were given to the evaluator with all identification removed and in random order. After the initial evaluation, the entire project was explained to the evaluator and he was asked to help fine-tune the APHID sites to match his preferred teaching style. He then re-evaluated the APHID sites and both evaluations were compared to the hand-constructed sites.

The effectiveness of APHID at creating reasonable hypermedia was further investigated by collecting simple metrics from a number of instructional web sites as well as from the sites created for the study. Those metrics were compared to determine if APHID creates hypermedia that is similar in size and construction to that created by instructional experts.

The efficiency of APHID was evaluated by comparing the amount of time taken by the instructional technology experts to build two web sites to the amount of time required to build two web sites using APHID. The remainder of this chapter explains this study in detail and presents the results from the evaluation.

5.2 Creating Evaluation Sites

The APHID process and software address the issue of organizing and presenting content in a hypermedia application and supplying appropriate hyperlinks for that content for a particular instructional purpose and user. Aspects of hypermedia creation that the APHID process does not address, such as writing textual bridges between ideas or concepts on a page, were not evaluated. To keep the focus of the evaluation on those aspects of hypermedia development that APHID was designed for, some of the activities normally associated with hypermedia development were restricted. The instructional technologist participants in the study were given the topic (plate tectonics) and the content to use in their applications. The topic of plate tectonics was chosen because it appears prominently in the Saskatchewan Science Curriculum Guide for public school students (Saskatchewan Education 1999) and the education students creating the sites were happier to participate when the exercise had potential future use as a classroom teaching aid. The instructional technologist participants were also given layout and visual presentation guidelines. However, they were allowed to organize the applications in any way they saw fit.

The first two work sessions were conducted as a group to provide instruction or additional help to study participants, if required. Instruction and guidance was provided by the computer science research assistant. Once it was clear that all participants were

comfortable with the creation of web pages, and that they understood the task, they were allowed to complete the creation of sites on their own time. All participants completed one web site suitable for a review of plate techtonics for advanced learners. Three participants also completed a web site suitable for introducing the topic to novice learners.

Of the two sites built by each participant, one was required to present the topic as it would be presented to immature students (early junior high) beginning their study of the topic. The other was to present it as a review to a more mature student requiring remedial teaching in the concept of Earthquakes. Half of the participants were asked to create the novice site as their first site; half created the advanced site as their first site. Each participant kept a log of time spent creating the web sites and completed a questionnaire concerning their experiences while creating the web sites. See Appendix D for the complete instructions given to participants.

Content was supplied to the participants in the form of text files containing groups of paragraphs organized by topic but not in any particular sequence. Each group of paragraphs was labelled with a heading indicating the concept. The participants were asked to use only the material from the files provided (by cutting and pasting) but were not expected to use all of it because far more was supplied than would have been reasonable to include in one web site. Participants were restricted to cutting and pasting materials to guard against different writing styles creating perceived differences in the quality of the web sites. Non-textual materials such as images were placed in a separate directory and referenced by file name in the text files and could be included in the web sites along with the selected textual items. The content for the sites was selected from a variety of educational web sites on geology and plate tectonics. The participants selected from the same set of content for each of the sites, varying the organization, the hyperlinks and the choice of which content to include and which to exclude. The participants were given a concept map as a reference for organizing their web-sites but were not required to necessarily follow its structure.

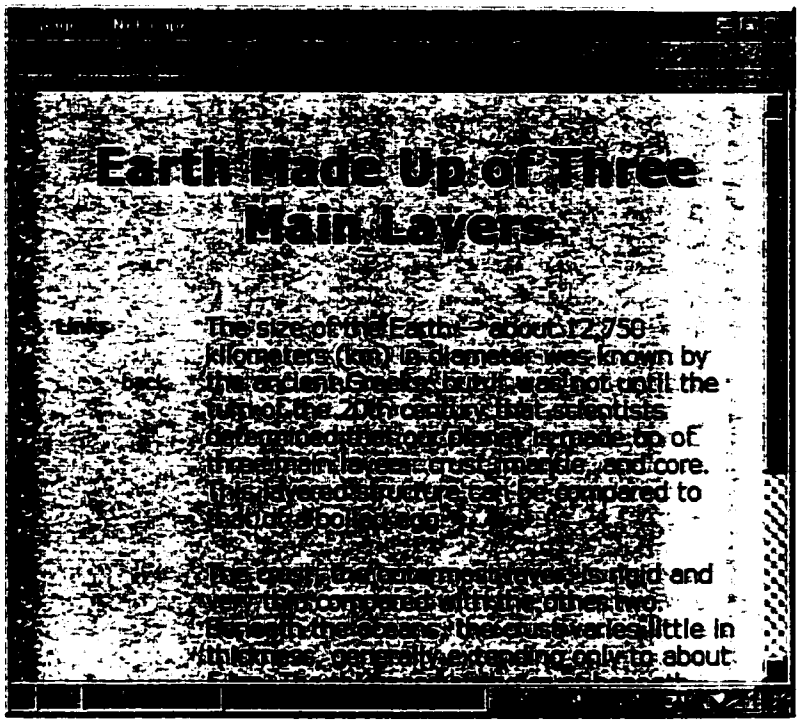


Figure 5-1 Example Page from Tectonics Application

The basic user interface requirements for the web sites (color, font, link placement and background) were controlled with a cascading style sheet (CSS) (Lie & Bos 1999) and a template HTML document. This controlled the appearance and layout for all the web sites produced because APHID does not yet have algorithms for selecting between different user interface attributes. Those aspects of hypermedia were not under consideration for the study and controlling for them simplified the task for the evaluator. An example of the layout of the hypermedia applications is shown in Figure 5-1. This particular page comes from one of the applications developed by the instructional technologist participants. The figure shows how the links are placed in a column on the left of the screen and data is placed on the right-hand side of the screen. It also shows the font and background selection.

Three web sites were created using APHID under the same restrictions as those imposed on the instructional technologist participants. The same guidelines were followed for user interface and link placement and content was taken from the same materials used by the instructional technologist participants. Two sites were created using APHID to

present material to novice learners and one site was created to present the material as a review for more advanced learners. The two novice sites balanced the total number of evaluation sites since three sites targeted towards novice learners were created by the instructional technologist participants, while four expert sites were created.

In the end, ten sites were created for the evaluation. Five of the sites were designed for novice learners and five were designed for advanced learners. Seven sites were created by instructional experts and three were created by APHID. Using style sheets, each site was normalized for font, color-scheme, and layout. The requirement that all information be cut/pasted from the supplied content controlled writing style. The differences between the sites were in the selection and organisation of pages, links and content.

5.3 Time Required to Construct Sites

The information presented when recruiting study participants included an estimate of the time commitment involved (10 hours) in an effort to reassure potential candidates that the time involved was not extraordinary as well as a promised monetary reward for completing the task. When people are given a target, they tend to work towards it, especially when they feel they are being rewarded for a specific amount of time. While none of the participants adhered rigidly to the ten-hour estimate all of them reported spending between ten and twenty hours in total building their web sites. Table 5-1 shows the times spent by the instructional technologist participants (labelled A, B, C and D). The most time spent on a single site was 10.5 hours, the most time spent in total was 19.5 hours. The minimum amount of time spent creating a single site was 4 hours.

Table 5-1 Time (in hours) Spent Building Sites

	A	B	C	D
novice site	9 (first)	4	10.5 (first)	N/A
advanced site	7	6 (first)	9	10 (first)

Creating the necessary models in APHID and using the software to create the three evaluation web sites took the author 9.5 hours in total. The human labour in creating hypermedia applications with APHID lies in creating the models and the data elements

since those activities must be completed before any hypermedia can be completed with the software. Model and data element creation consumed slightly over 9 hours of time for the evaluation applications. Approximately one of those nine hours represents time lost due to the fact that the software was prototypical and lacked suitable mechanisms for creating data elements. For instance, the XML representation of the data elements had to be created by hand using a text editor and APHID accessed the file system containing the XML data elements to build the web sites. APHID kept meta-data about the data elements including filenames and paths. Occasionally filenames were transcribed incorrectly and errors would occur in the processing because files could not be found. These types of delays would not happen if APHID were not a prototype system. Once the models were created, different hypermedia applications could be created in less than five minutes. The concept map created in APHID for the evaluation is shown in Figure 5-2.

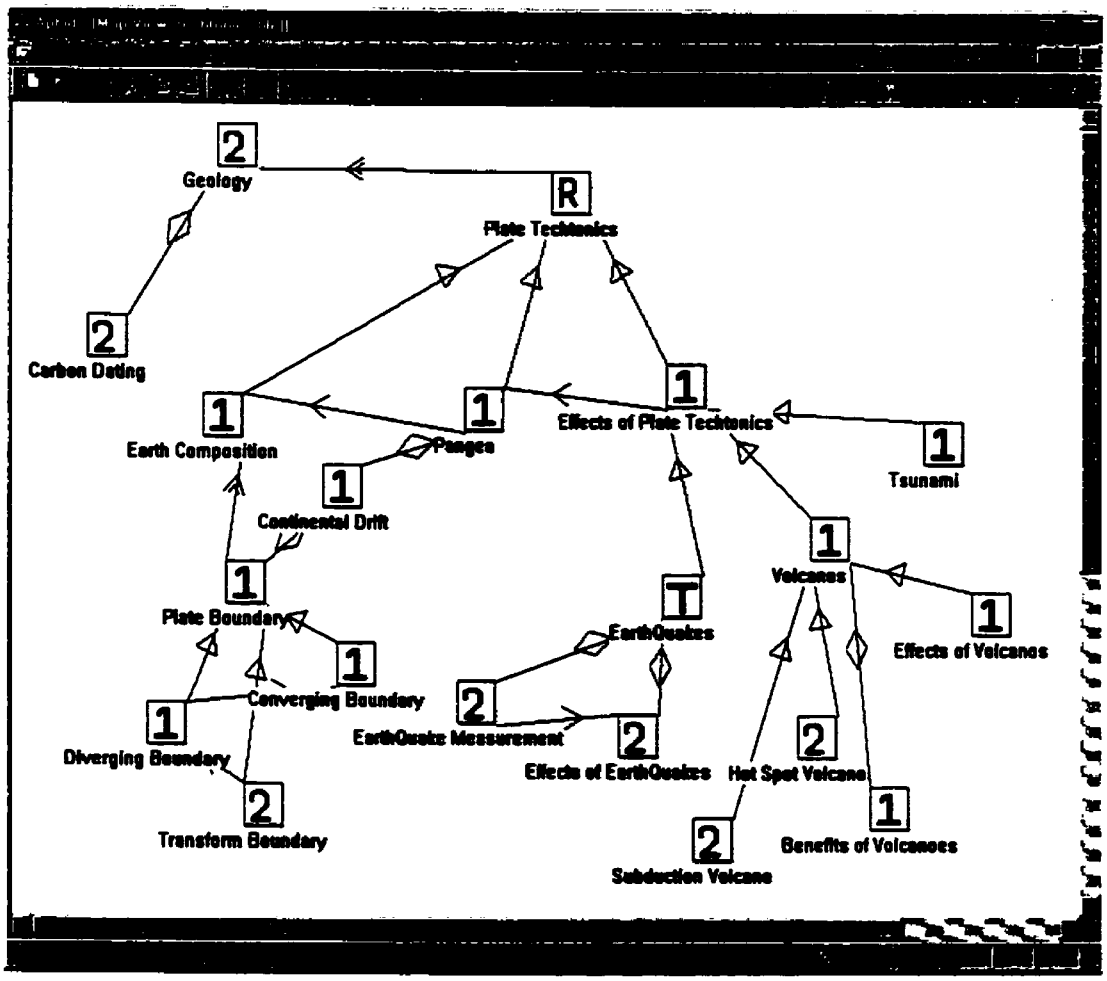


Figure 5-2 APHID Concept Map for Evaluation Sites

In total, 157 data elements were created (as individual XML files) for use with the APHID web sties. Table 5-2 shows the activities associated with building hypermedia applications using APHID and the approximate amount of time required for each activity while constructing the evaluation sites.

Table 5-2 Time Required to Construct APHID Sites

Activity	Time Required
Creating Concept Map	1.5 hours
Creating Data Elements	6 hours
Linking Data Elements to Concept Map	1.5 hour
Selecting Site attributes (3 sites)	10 minutes
Generating web site (3 sites)	20 minutes
Total	*9.5 hours*

The instructional technologist participants in this study were asked to complete an exit survey to determine their thoughts on the process. They were in agreement that the notion of creating instructional web sites with a particular instructional goal was important and felt that some aspects of the process could be automated. Suggestions for automation included indexing content, formatting and creating navigational elements. They indicated that the creation of the second web site was easier than the first one, although no one took significantly less time to create their second site (except for the person who did not create a second site). Appendix E gives a summary of the comments given by the participants.

5.4 Site Size and Organization

Each participant had a slightly different approach to meeting the objectives given them constructing the sites. Each did provide some notion of sequence within the site, some by using *next* and *back* links and others by using an ordered index page. Generally, the sequence of presentation followed the hierarchy of the data model (as represented by the concept map the developers were given), starting with general concepts and expanding into more specific concepts. A single concept was explored in depth before going on to a second concept, an approach similar to the Depth First Instruction pattern provided by APHID. This seemed to be the approach taken regardless of whether the site was intended to be remedial site or an introduction.

No single participant-developed site covered all of the 53 concepts suggested by the original concept map. The differences between the sites for novice and advanced learners seems mainly to be in terms of the content selected and in the concepts covered, not in the order of presentation nor in the number or order of hyperlinks on individual pages. Because the participants were not required to use the concept map, neither a numerical comparison of the concepts covered nor an in-depth exploration into the method used to determine the order of presentation is possible.

The computer science research assistant collected measures for the ten study web sites to compare the sizes of the sites. The measures collected were:

- The number of web pages in the site (a page is defined as a single HTML file)
- The number of words in a each web page
- The number of concepts covered in the site
- The number of data elements used to create the site

It is simple to count the number of pages and words for a web site using an automated script. Each of the web sites under consideration was contained in a separate directory, so the script simply counted the number of files to determine number of pages, and then counted the number of words in each file after stripping the HTML code.

To determine the number of concepts, the research assistant was asked to determine which concepts from the example concept map were covered in the web site. He simply kept a list of concepts covered as he examined each site and compared the items on that list to the concepts represented in the original map.

It is more difficult to ascertain precisely how many different data elements are used since a formal definition for data element is not available. For the purposes of these measurements a data element was defined as a unit of information about a particular concept. It was further specified that a typical data element could consist of a paragraph, picture, or set of related paragraphs that were cut/pasted from a single source. The analyst counted the number of data elements in each site by visually inspecting each site. The measures collected are given in Table 5-3, while a comparison of the averages for APHID created sites vs. hand-created sites is given in Table 5-4. The first table also

shows with which instructional pattern the three APHID sites were constructed. This information is important because APHID varies the number of concepts and elements according to the constraints associated with the selected instructional pattern.

Table 5-3 Measurements Taken from Evaluation Sites

	Creator	Build Order	webpages/ site	Words /page (avg)	concepts	# Data Elements
Novice Sites						
Site 1	A	1	43	245	18	58
Site 2 (spiral)	Aphid	N/A	50	131	19	78
Site 3	B	2	30	277	11	42
Site 4	C	2	13	412	12	47
Site 5 (depth)	Aphid	N/A	19	289	19	95
Averages:			*31*	*270.8*	*15.8*	*64*
Advanced Sites						
Site 6	D	1	14	661	13	56
Site 7	C	1	13	411	16	75
Site 8 (remedial)	Aphid	N/A	23	349	20	152
Site 9	B	1	13	441	19	80
Site 10	A	2	28	278	10	57
Averages:			*18.2*	*428*	*15.6*	*84*

Table 5-4 Measurement Averages

	Aphid averages	Hand-created averages
# webpages/site	34.5	22
#words/page (avg)	210	389.2857
# concepts	19	14.14286
# Data Elements	86.5	59.28571

The measures show that, on average, the APHID-produced sites covered more concepts than the hand-created sites, and used more data elements in the process. Unlike the hand-crafted sites, the number of data elements used by APHID for an advanced learner was significantly more than the number used for a novice learner.

Even though APHID used more data elements, the average number of words used per page is similar for most of the sites and on average APHID uses fewer words per page. Generally more words per page were used by both APHID and the instructional designers when creating a site for advanced users. APHID used neither the most words

per page nor the fewest words per page with the exception of the site using the Spiral Curriculum pattern (Site 2) which does have the fewest average words per page. This is consistent with the goals of the spiral curriculum pattern, which is to present material in small sections and to repeat concepts frequently. Given that none of the hand-produced sites used a spiral-like approach, it is reasonable that none of them have the smaller number of words per page.

There is a small difference when the numbers of concepts covered are compared. APHID covers more concepts for both the novice and the advanced sites. The difference is not large however, and indicates that the instructional designers who were creating sites by hand and the instructional designer working with APHID had similar notions about what should be included. Figure 5-3 is an example page taken from the APHID site using the Spiral Curriculum pattern. A comparison of it and the page illustrated in Figure 5-1 shows that similar information is presented in both pages. Generally, the sites produced by APHID were similar in size and content to the sites created by the instructional designers.

To examine the possibility that APHID sites were similar to the others only because the content was the same a second set of measurements were taken. Undergraduate university students collected simple metrics (as part of a university class assignment) from ten different web sites featuring tutorials intended for adult learners. The students were asked to count the number of pages in the web site, the number of concepts covered in the web site, the number of words on each page and the number of data elements on each page. A definition was given for concept and data element, but the identification of such was left to the students' discretion. Students were asked to take measurements for two of the ten possible sites, but were allowed to select the two sites thus the ten sites were not represented evenly in the sample. A compilation of the results from this assignment is presented in Table 5-5. The table shows the mean counts for an entire web site.

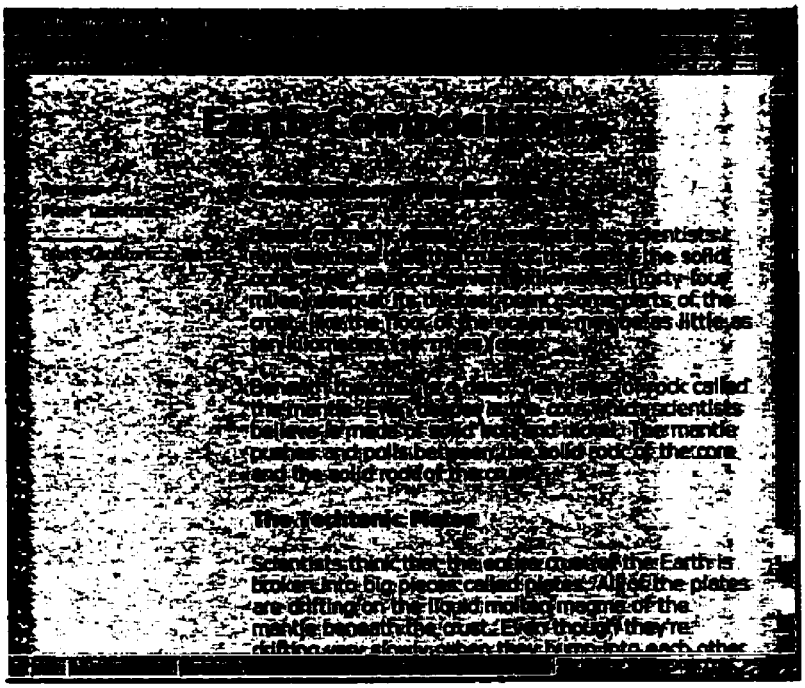


Figure 5-3 Example of APHID Produced Page

Table 5-5 Measures from Student Assignments

	#times in sample	#pages /site	#words/ page (avg)	#concepts	# Data Elements
www.cs.usask.ca/classes/3671/projects/99 group7/	10	22.3	481.3	17.1	90.73
www4.ibm.com/software/developer/education/xmlintro/	25	54.8	68.7	32.2	209.81
www.cs.usask.ca/classes/371/projects/99group9/	8	58.9	276.3	76.0	712.81
/www.cs.usask.ca/classes/371/projects/99 group3/	24	21.2	400.6	43.3	131.82
Pdbeam.uwaterloo.ca/~rlander/XML_Tutorial/xml_tutorial.html	13	12.9	420.8	12.9	52.37
www.cs.usask.ca/classes/371/projects/99group8/	10	24.6	331.2	23.0	100.93
www.level2.com.au/xml/3.htm	20	6.8	403.7	9.1	25.84
www.cs.usask.ca/classes/371/projects/99group6/	9	51.3	200.9	46.4	183.31
www.wdvl.com/Authoring/Tools/Tutorial /index9.html	2	5.0	851.3	5.0	36.00
Averages		*28.6*	*381.6*	*29.4*	*171.51*

Sixty-seven students submitted measures for two web sites each. One of the web sites that was given to the students was simply one long file, and didn't fit the notion of tutorial used for this research. The measures collected for that web site are not shown in the table. The table shows the mean counts for an entire web site. Even allowing for measurement error, which may be quite high because of the inexperience of the measurers and the lack of precision of the definitions, the measurements indicate general trends in tutorial sizes. The web sites analysed by the students used a similar number of pages, concepts and data elements to APHID. The web sites tended to have more words on a page than APHID sites, which is reasonable considering these sites are intended for adult learners. The indication is that APHID produces sites that are comparable in size to those produced by other authors.

5.5 The Evaluation by the Independent Evaluator: Experimental Setup

An independent evaluator, with experience both in instructional design and in web site construction, was recruited to rate each web site, i.e. those produced by APHID and those produced by the instructional technologist particpants. Prior to the evaluation, the

evaluator was unaware that any of the sites had been machine-created, or precisely what the purpose of the evaluation was. He was told that the evaluation was to examine different methods of creating instructional hypermedia (see Appendix F for the complete set of instructions given to the evaluator). The web sites were given to him in random order with no possibility of identifying authors.

The evaluator was given a set of criteria to consider when rating the sites and was asked to rate each site for each criterion. The rating criteria were selected to focus the evaluation on web site structure, organization and content selection. The set of criteria is a compilation of suitable entries from a variety of web site ratings instruments (Rettig 1996; Joseph 1999; Kotlas 1999). The set of criteria, along with the descriptors provided to the evaluator is given in Figure 5-4. The first nine criteria were rated using an ordinal scale ranging from one to ten, where one represented a poor effort and ten represented an excellent one. The last four criteria were also rated on an ordinal scale, but the meaning of the scale was altered so that one represented too few items, ten represented too many and five represented the correct number of items. This was done because those criteria were dealing with quantities that were difficult to describe using a simple 1-10 scale. The evaluator was asked also to provide written comments whenever possible. His evaluations and written comments are provided in Appendix G.

1. **Navigability**
The learner can move from page to page and item to item with ease and without getting lost or confused. Links are well organised and appropriate link choices are easy to find.
2. **Suitable Hyperlinks**
The links on a page lead to appropriate pages for the learner to select as their next page. Any choice of hyperlink on a page results in a logical ordering of the material.
3. **Organisation of Hyperlinks**
Hyperlinks are organised within the page to help the learner make suitable choices.
4. **Instructional Strategy**
The site has a clear instructional plan. Enough ordering supplied within the hyperlinks and pages to execute that plan. It is as easy, or easier, to use the site as an instructional tool as it is to use it for free exploration. The material presented is not overly repetitive (given the learner's level).
5. **Order of Presentation**
The order in which concepts are presented is suitable for the particular instructional treatment of the material. (This applies more to the instructional presentation, if one is present)
6. **Organisation of Material (individual pages)**
The information is clearly labelled and organised in a reasonable fashion. Difficult material is broken into manageable chunks.
7. **Suitability of Content**
The information is at a level appropriate to the learner's level (understandable, but challenging enough to promote some thought and reflection).
8. **Structure of Topic Emerges**
A sense of the overall structure of the topic emerges as the learner traverses the site. After using the site, a learner would be able to construct a model of the topic (concept-map, outline, etc). (Assuming they read and understand everything)
9. **Linkages between Concepts**
Concepts in the topic are connected (via hyperlinks) to reflect the whole topic correctly. For instance, if one was learning about cars, and the sub-concepts were wheels, engine, and pistons, there should be hyperlinks between cars and wheels, cars and engine and engine and pistons. It is less likely that there would be a hyperlink between cars and pistons.
10. **Quantity of Pages**
The quantity of pages in the site is neither too few nor too many.
11. **Quantity of Hyperlinks**
There are sufficient links for both backward and forward movement. There are enough links to allow learners to explore the topic and discover the structure of the material.
12. **Quantity of Material (entire site)**
The topic is adequately covered. The site has a suitable amount of material for the level of the learner.
13. **Quantity of Material (individual pages)**
14. Pages have enough material on them, without being overwhelming.

Figure 5-4 Criteria Used forEvaluation

The criteria were selected to identify strengths and weaknesses in the organization of the web sites. Four of the criteria (1, 2, 3 and 11) are concerned with the hyperlinks found in the web site and the over all structure of the site. They elicit comment about the overall navigability of the site, the appropriateness of the hyperlink targets and the organization and quantity of the hyperlinks. Four of the criteria (4, 5, 8 and 9) deal with the instructional nature of the hypermedia applications and attempt to determine if the

web site is organized in a way suitable for instruction about the topic of plate tectonics. The remaining five criteria address issues about the content of the web site including quantity, appropriateness and organization. Together they provide a reasonably broad view of the quality of a web site.

The second part of the evaluation was to experiment with APHID's ability to accommodate the personal preferences of instructors (and users). Once the evaluator had rated each of the web sites he was told about APHID, shown which sites were constructed by APHID and given the opportunity to request fine-tuning the APHID sites to better conform to his preferences. He was given a complete explanation of the project and invited to observe how APHID is used to create web sites. An interesting note is that the instructional patterns and their descriptions proved to be quite valuable in describing the process of building different types of hypermedia sites with APHID. Based on his initial evaluations, two of the web sites he had evaluated were selected for rebuilding using attribute values suggested by the evaluator. He then re-evaluated those two sites using the same criteria.

5.6 The Evaluation by the Independent Evaluator: Results

The results from the evaluator's initial evaluation of the web sites are shown in Table 5-6. One of the original thirteen criteria has been dropped from the results because an interview with the evaluator revealed that there had been a misunderstanding about its definition. The definition of Instructional *Strategy* assumed by the evaluator was quite dissimilar from the intended definition, consequently the ratings for that criteria were of no value to the discussion. As mentioned previously, the first eight categories in the table are rated on a scale from 1 to 10, where 10 is the most desirable score. The last four categories are rated on a scale from 1 to 10 where 1 represents too few, 5 is a desirable score and 10 represents too many.

The columns in bold type contain the ratings for the sites created by APHID. None of the web sites evaluated received excellent (a score of 8 or higher) ratings in all categories and all of the web sites evaluated received excellent ratings in some

Table 5-6 Raw Evaluation Results

	APHID sites			Hand Crafted Sites						
Site #:	**2**	**5**	**8**	**1**	**3**	**4**	**6**	**7**	**9**	**10**
Rated on a scale of 1-10 where 1 is poor and 10 is excellent										
Navigability	1	2	3	3	7	8	9	4	7	6
Suitable Hyperlinks	4	3	3	7	7	9	9	4	7	6
Organisation of Hyperlinks	2	2	2	3	7	9	8	2	7	6
Order of Presentation	2	5	5	7	5	6	4	3	5	4
Structure of Topic Emerges	1	6	4	2	6	6	5	5	6	5
Linkages between Concepts	2	6	4	3	6	6	3	4	6	4
Suitability of Content	9	6	6	3	1	7	7	4	6	7
Organisation of Material (individual pages)	8	5	4	6	2	6	3	4	2	6
Rated on a scale of 1-10 where 1 is too few, 10 is too many, 5 is just enough										
Quantity of Material (entire site)	7	5	7	9	4	2	4	5	4	5
Quantity of Pages	7	5	3	8	6	2	4	5	6	5
Quantity of Hyperlinks	1	4	9	3	7	5	1	4	7	4
Quantity of Material (individual pages)	7	5	8	9	6	5	5	5	6	5

categories. In a post-session interview, the evaluator stated that he had a personal preference for building and teaching with web sites that were hierarchical in nature. He felt that he may have rated sites that were hierarchical as being more desirable than sites that were constructed using a different model (like the spiral curriculum).

Table 5-7 provides a more concise view of the data. To simplify comparisons, the scores for the last four criteria in Table 5-6 (representing quantities) have been transformed from the too-few, too-many scale to a 1-10 scale using the following formula: N_t=10 - Abs(N-5)*9/4 where N is the original score given on the criterion. This formula was used to transform the results because the original scale used didn't have an integer middle value. Table 5-7 contains these transformed scores as well as the other raw scores for the APHID sites. The criteria are grouped according to the type of information elicited by individual criteria. Comparing the means with the scores given to individual APHID-produced sites yields some interesting information about the strengths and weaknesses of APHID from the viewpoint of the evaluator. The APHID sites were usually rated well in criteria related to content selection and quantity (with one exception) and less well for criteria related to navigation.

Table 5-7 Mean Scores

		APHID sites				Hand crafted sites
	Site #:	2	5	8	Mean	Mean
Hyperlinks and structure	Navigability	1	2	3	2.00	6.29
	Suitable Hyperlinks	4	3	3	3.33	7.00
	Organization of Hyperlinks	2	2	2	2.00	6.00
	Quantity of Hyperlinks	1	8	1	3.33	6.14
Instructional nature of application	Order of Presentation	2	5	5	4.00	4.86
	Structure of Topic Emerges	1	6	4	3.67	5.00
	Linkages between Concepts	2	6	4	4.00	4.57
Content of application	Organization of Material (individual pages)	8	5	4	5.67	4.14
	Suitability of Content	9	6	6	7.00	5.00
	Quantity of Pages	6	10	6	7.33	7.11
	Quantity of Material (entire site)	6	10	6	7.33	6.79
	Quantity of Material (individual pages)	6	10	3	6.33	8.07

An unanticipated difficulty arose from differences in how the instructional designers interpreted the rules given them and how the same rules were applied to the APHID-produced sites. One example of this was the use of next and back as hyperlink labels. The instructional designers added the words next and back in as link labels but APHID used the titles of the pages as link labels because next and back weren't explicitly given as part of the text. APHID used color to distinguish between types of links rather than explicit labels. The evaluator understandably found the links labelled next and back to be clearer and rated the sites with explicitly labelled next and back links higher in several categories. Another example is in the use of off-site hyperlinks. Hyperlinks to other sites were not intended to be part of the evaluation web sites, yet some of the participants included such hyperlinks and none were included in the APHID-produced sites.

Many of the evaluator's comments on the APHID sites were concerned with link labels and hyperlink arrangement. In general the features he noticed arose from strict adherence to the rules given or simply from different understandings of how the rules

should be interpreted. Fortunately, the concerns he raised were easily addressed when he fine-tuned the APHID sites to his personal preferences.

One particular attribute of two the APHID sites caused low ratings in several categories. The evaluator did not feel that the Spiral Curriculum pattern was one that he would ever use, and rated sites that demonstrated the pattern low on several categories (which affected one entire novice site and a portion of the remedial site). He disliked the fact that the same concept was presented in several segments although he liked the amount of information that was on individual pages and the total quantity of information. He suggested that he would have liked it better if the titles of the pages were less similar. When APHID creates a second page for a particular concept (for instance, if the concept of Volcanoes were broken into three parts, there would be three separate pages for it) it uses the same title for the page but adds a *part N* label. This was done to avoid the need to generate text dynamically within APHID and to ensure that the APHID sites used titles that were contained in the original materials.

The evaluator consistently rated the APHID sites below average in navigability, organization of hyperlinks, and suitable hyperlinks. These categories were intended to address the navigability of the site, the suitability of target pages for linking from the source page and the organization of hyperlinks within an individual page. The comments from the evaluator (both written and verbal) indicated that he evaluated the hyperlink text and formatting for all three categories. He felt that the style used by APHID (colored link-types) was inappropriate and that the *part 1* and *part 2* text generated by APHID to differentiate between different versions of pages was confusing. Consequently, he rated the APHID sites lower for these categories. The evaluator suggested that labelled categories of links would have been helpful which was a presentation feature that had been considered and not implemented because it was not in conformance with the directions given to the educators who created the other sites.

A mixed rating was given to the APHID sites for the *quantity of material* category. The sites created for novices were rated well for that category but the site created for the advanced learner was not. The evaluator's comments about this category indicated that

he had a different mental model of advanced learners than was used to create the sites. He felt that there was simply too much material presented at once, even for what he felt would be an advanced learner. Quantity of material is an attribute that is easily set within APHID, which made it simple to address his concerns about too much material.

The APHID produced sites were consistently rated near or above average for several other categories. The evaluator felt that the software consistently chose suitable content, organized the individual pages well and created a suitable number of pages with a suitable amount of content. Given that the evaluator stated that the Spiral Curriculum pattern was not something he would choose to use, it is interesting to compare just the Depth First and Remedial sites with the averages for the hand-crafted sites. Both the Depth First site and the Remedial are also rated near or above average for structure of topic, linkages between concepts and the order of presentation.

In summary, the evaluator felt that APHID created sites that were near or above average for seven of the twelve categories. The average rating for APHID sites was near or above the average rating for most categories dealing with instruction and content. It is significant that, prior to being told, the evaluator was not aware of, and did not suspect, that some of the sites were created automatically. His concerns about the APHID sites in the other categories were a result of using default values for attributes designed to be have values assigned by the instructional designer who would use the sites.

The default values for the APHID sites were chosen by the author to conform to personal preferences and assumptions about the target learners. The terms novice and advanced learner are general terms, and even the understanding of what a junior high student might be capable of differs from educator to educator. The default values selected for the APHID sites were inappropriate for the evaluator's preferences for a number of reasons. First, the evaluator was familiar with a different curriculum than the author, so the content selected for the sites would normally be taught to older students by the evaluator. This meant that he felt that in general, the content would be difficult for the target age-group. A second reason was that the evaluator was imagining using the sites with a class of learners rather that with individuals and he did not feel that the

APHID produced sites were suitable for an average student. Because he did not realize that the sites had been machine-produced, he couldn't imagine going to the effort of making a web site for an individual learner.

A third reason that the defaults didn't seem appropriate to the evaluator was a result of differing understandings of what learners are capable of. The author has worked with different types of students than the evaluator and felt that average students were more capable and more attentive than did the evaluator.

Similar differences in instructional preferences will appear given any two educators. APHID makes accommodation for these differences by allowing educators to override the default settings that govern content selection, link selection and organization of pages.

5.7 Fine Tuning and Re-evaluating the APHID Sites

The flexibility of the APHID approach was demonstrated when two of the APHID sites were re-built using the evaluator's preferences for site and page attributes. The evaluator was invited to select two of the sites that he felt best reflected how he might teach the topic for fine-tuning. He selected site 2 and site 8 for fine-tuning. The possible settings were explained to him and he selected those that he felt would improve the web sites.

The evaluator chose to reduce the number of data elements presented to both novice and advanced learners. He also chose to reduce the number of hyperlinks allowed on a page. A different presentation for hyperlinks was selected to include a link-label as well as the link title. He was satisfied with the ordering of data elements and the selection of concepts so those settings were not altered. The number of parts a concept could be broken into was reduced to two (from three).

Even though these changes required modification to the source code for APHID and a recompilation of the program (which was an artifact of the prototype and would not be the case in a production-quality system), the total time required to make the changes was less than half an hour. This includes time for making the changes, recompiling the

Table 5-8 Re-evaluation Results

		Aphid sites			Hand Crafted sites
		8	2	Mean (2 sites)	Mean
Hyperlinks and structure	Navigability	8	8	8	6.29
	Suitable Hyperlinks	8	8	8	7.00
	Organisation of Hyperlinks	7	7	7	6.00
	Quantity of Hyperlinks	10	10	10	6.14
Instructional Nature of Application	Order of Presentation	7	7	7	4.86
	Structure of Topic Emerges	10	7	8.5	5.00
	Linkages between Concepts	7	9	8	4.57
Content of Application	Organization of Material (individual pages)	7	7	7	4.14
	Suitability of Content	7	8	7.5	5.00
	Quantity of Pages	8	10	9	7.11
	Quantity of Material (entire site)	8	10	9	6.79
	Quantity of Material (individual pages)	10	10	10	8.07

source code and regenerating the sites. An additional half-hour was spent familiarizing the evaluator with the APHID process.

The evaluator performed a second evaluation on the revised web sites, the results of which are shown in Table 5-8. As can be seen from the table, the evaluator felt that the deficiencies he had noticed in the web sites had been corrected by changing parameters to reflect his preferences. He indicated that both sites, in their revised form, were clearly above average. What is interesting is that changing relatively few parameters made such a significant difference to the evaluator's perceptions of the sites.

Overall, the portion of the study involving the evaluator shows that instructional patterns with default attributes can be used to create acceptable hypermedia applications, and that the APHID approach to realizing those patterns can be quickly and easily tuned to create good hypermedia applications that appeal to specific users.

5.8 Summary

We have shown that APHID can be used to create customized hypermedia applications that are as good as those created by hand and that the process of creating those

applications is more efficient than creating similarly customized applications by hand. Using APHID to create hypermedia applications is more efficient if the intention is to create more than one application from the same information. It is not necessarily more efficient (although it may result in better hypermedia) if only one application is desired. We have also shown that at least a small sample of educators sees a need for web sites that are customized to particular instructional purposes which points to a need for creating multiple sites from the same information.

Simple metrics collected indicate that APHID creates sites roughly comparable in size and complexity to other instructional websites. APHID provides a similar (or slightly greater) depth of coverage in terms of number of concept covered in a detailed web site and the number of data-elements used.

The idea of semi-automatically creating hypermedia applications is appealing. All of the study participants indicated that they felt parts of the hypermedia creation process could be automated. All participants also thought that learners would benefit from web sites that were specifically tailored to their needs. The evaluator also felt customization would be beneficial, but indicated that he would never have considered actually creating individualized sites for particular students because of the time involved in creating multiple sites. He was most positive about an automated approach to individualization. APHID makes customization for learners possible because it is more efficient than creating sites by hand. Building a single site using APHID takes approximately the same amount of time as creating one by hand. Creating additional sites from the same materials with APHID takes significantly less time than creating additional sites by hand.

APHID is not intended to be a completely autonomous generator of hypermedia. The intent is that an instructional designer can use the method and the software to easily create instructional hypermedia applications that are tailored to meet the needs of specific users or to teach using a specific strategy.

The initial evaluation of the web sites showed that, even when using the default attributes for the instructional patterns provided with APHID, the resulting web sites

were acceptable. In fact, the APHID sites were rated near or above average for most categories. The power of the APHID approach is that it provides defaults for designers who are inexperienced (or in a hurry) and allows fine-tuning by those designers who wish for more control. The re-evaluation of the APHID sites after the evaluator was allowed to customize the sites demonstrates that APHID can produce hypermedia that meets the personal goals and requirements of individual instructors resulting in well-designed applications that are also flexible and maintainable.

Chapter Six

Conclusions and Future Directions

The previous chapter illustrated that this research has achieved the objective set in the beginning. We have shown that the APHID approach to designing and producing instructional hypermedia can produce satisfactory hypermedia applications more efficiently than humans are able to.

The first chapter of this thesis suggested that the achievement of that objective had the side effect of realizing five additional goals. This section explores those additional goals and examines how they have been accomplished.

To reiterate, the five general goals stated in Chapter one were:

1. To identify common components within a hypermedia application intended for instructional use. These components will be comprised of element types and relationship types. Any instructional application is composed of elements and relationships that are drawn from this set of common components.
2. To identify the current 'best practices' for the creation of instructional hypermedia applications and to codify those practices in the form of patterns. Some patterns are design patterns, some are instructional patterns through the conceptual space. Each instructional pattern corresponds to a specific instructional strategy or goal.
3. To identify a development approach for hypermedia that utilizes patterns and concept maps to produce well designed instructional hypermedia applications. The production of the applications is partially automated.
4. To implement a prototype software system that embodies the APHID approach.

5. To evaluate the implemented system to demonstrate that the research objective has been met.

The first goal of identifying common components has been met, although certainly the list of components identified is not exhaustive. We have used data elements, a variety of instructional classes and several types of relationships to construct a wide variety of hypermedia applications. Different types of components will doubtless be identified in the future.

We have likewise identified and recorded a small sample of patterns for the development of instructional hypermedia. The sample proved to be enough to accomplish the objective set for this research, but does not include many alternate approaches to instructional design. This goal proved to be the least simple to realize because there is little agreement among educators as to what comprises good instruction or good design. Much of the future work on the APHID approach must focus on identifying additional instructional patterns and codifying them.

The remaining three goals have clearly been accomplished. That the applications produced by APHID are well designed has already been established. The APHID approach uses concept maps to describe the domain of instruction and patterns to describe both the presentation of instruction and the process of instruction. APHID provides partial automation in the prototype implementation and more automation is likely in future versions.

The process of achieving these goals and the primary objective has led to some interesting observations and contributions to the area of instructional hypermedia design. The next section explores those research contributions.

6.1 Research Contributions

The main research contribution of this work is the development of a method for modelling and creating instructional hypermedia systems. The method developed is based on sound software development principles and on principles of instructional design. It also provides scaffolding for instructional designers who may not have

knowledge about hypermedia construction as well as for software developers who may not have knowledge about instruction. The APHID approach to constructing instructional hypermedia provides a means to create instructional hypermedia applications tailored to the instructor's purpose and preferences. The underlying assumption is that the instructor knows best the needs of the intended learners and is in the best position to provide the bounding information for the APHID software.

The key difference between this research and other approaches to hypermedia design is the inclusion of the semantic model as one of the design models for hypermedia systems. Semantic modelling often requires detailed knowledge about the instances that make up the domain. For instance, consider the models created during indexing of a large body of text. That process requires a large quantity of domain knowledge to disambiguate words and identify synonyms. Semantic modelling is frequently deemed to be inappropriate for product analysis and design. The APHID approach to semantic modelling with a concept map creates a middle ground between the domain model, which is quite general and the instances, which are always specific. The knowledge used in this model is contained in the relationships allowed in the concept map. The concept map is general enough to be reusable for other applications in the same domain of instruction, and the patterns of instruction can be reused in other domains, provided similar types of relationships exist between the concepts.

The APHID approach to instructional hypermedia design has two key differences from the approach taken by designers of intelligent tutoring systems. The first difference is that the APHID domain models do not have a target or goal concept where intelligent tutoring systems usually assume that one concept of the domain model is the goal and that the steps necessary to reach the goal are precisely identifiable. Instead, the goal in an APHID-produced hypermedia application is to present an entire subset of the domain model to a learner in some predictable, organized fashion. APHID applications are designed to support the human instructor rather than to provide direct instruction.

APHID's roots in software engineering methodology provide the second difference between it and approaches to intelligent tutoring system design. APHID allows the

calculation of metrics for each hypermedia application. The metrics can be used to constrain the application size and structure, resulting in an application that conforms to the current understanding of how good hypermedia should be structured.

The research makes a contribution towards the use of patterns in analysis and design of hypermedia. Patterns have been applied to many different design problems by a variety of researchers. The contribution of this research to the patterns community is the use of patterns to describe instruction, and subsequently to guide the development of the hypermedia application. Even though relatively few instructional patterns are identified as part of this research, the concept is novel and has been a useful tool for communicating with educators during this research. The use of patterns for communicating instructional ideas should simplify the process of identifying suitable organizations for different types of instructional hypermedia.

Although the focus of the work is on instructional hypermedia applications, any hypermedia application for which the designer has a clear 'probable path' in mind could be developed using the APHID approach. APHID provides a solid baseline for future research into analysis and design of any sort of targeted hypermedia applications. A targeted hypermedia application is one for which the author has a single, specific purpose in mind. Instruction is one type of targeted application, but many others exist. The advertising industry creates targeted media, as do motivational speakers, industrial trainers and designers of help systems. This work has shown that it is possible to create good hypermedia applications algorithmically, the next step is to explore how to improve those algorithms and to apply them to similar domains. The next few sections of this chapter discuss future possibilities for APHID including some possible improvements and some ideas for how APHID could be used in arenas other than instructional design.

6.2 Improvements to APHID

There are potential improvements to the APHID software that arise simply from the fact that it is a prototype system and is not built to be either robust or efficient. These

improvements include the re-design of many of the key classes, the addition of robust error checking on data, and the creation of an XML editor to simplify data entry. These types of alterations will not change how APHID works or improve its ability to create hypermedia, they simply improve the way the software is written.

There are several other possible improvements to the design of APHID that would add to the functionality or feature set of the system. Generally the improvements are one of two types: alterations or additions to the instructional and presentation patterns, or alterations and additions to the algorithms used to generate the hypermedia applications. Most of the improvements suggested can only be realized after significant further research into related disciplines. The next two sub-sections explore these two types of improvements.

6.2.1 Additional Patterns

Two types of patterns are used within APHID to help describe hypermedia applications, instructional patterns and presentation patterns. The patterns used for the prototype software were constructed based on the instructional experiences of the researchers. A researched approach to creating additional patterns could improve the types of applications that APHID creates.

New presentation patterns should be derived from information about how learners work with instructional hypermedia and about which features are most commonly useful to learners. Such work could determine, for example, the preferred ordering for different types of instructional elements, answering questions like "should the definition go ahead of the example or after the example". This same research could yield information about the optimal values for settings that control the size and shape of the hypermedia, like the number of elements and the number of hyperlinks. This research would have to take into account the abilities and preferences of a variety of learners and would likely also result in the definition of a larger variety of learner types for APHID.

New instructional patterns should be developed after a methodical investigation into what kinds of hypermedia educators create and how effective those sites are. The

research also needs to examine classroom teaching practises and the current understanding of instruction in general. Many teaching strategies exist that could be modified for hypermedia presentation, but presently most hypermedia seems to be strictly hierarchical.

One requisite for the development of new patterns (both instructional and presentation) is an understanding of the characteristics of hypermedia and an examination of which of those characteristics are known to be of value instructionally. Presently there is no consensus on how to describe a hypermedia application by measuring it. Depth, breadth, number of nodes and similar measurements all seem to be likely candidates, but there is no understanding of how the different values for those measurements affect the usability of the hypermedia application. One method of determining characteristics of sites would be to use the types of measures for hypermedia discussed in Section 2.2.3.3. An interesting experiment would be to juxtapose such measurements against student and instructor ratings for a variety of instructional sites and determine which characteristics seem to be most important to users. One promising source of information about how usability and structure of web sites are interrelated is the user studies presently being done in conjunction with the MetaLinks project (Murray, et al. 1999)

One user interface construct that is necessary for the APHID hypermedia is an overview. The overview could be a map, an omnipresent index, a fish-eye view or any of a number of other constructs. The investigation into additional presentation patterns should consider the necessity for supplying a context to learners and developing a pattern that accommodates that need.

6.2.2 Additional and Improved Algorithms

APHID uses a variety of algorithms to create a hypermedia application. The software is designed to allow new algorithms to be easily incorporated into the design, which facilitates the creation of new types of hypermedia applications. There are also points in the prototype software where the existing algorithm is sketchy or experimental and should be refined in future versions of the software. Two points in the prototype

software are in need algorithmic refinement: constraint management and data element selection. New algorithms should be explored for traversing the concept map and the use of XML and XSL should be further incorporated into the program design.

APHID maintains a set of rules for the construction of hypermedia applications, but it has no facility for true constraint solving. As the rule-set becomes more complex, simple application of the rules is unlikely to suffice because conflicts will arise. Rules are applied as nodes are created in APHID, and should be checked each time a new node is created and, if necessary, modifications should be made to the navigation model to ensure adherence to the constraints. Presently rules are relaxed rather than employing a backtracking mechanism to make different choices.

An example of this type of problem arises when the maximum depth from the root node is exceeded. Ideally, APHID would backtrack and make the parent node of the violating node more detailed so that the critical information of the 'out of depth' node was included in its parent. Presently the violating node is simply deleted and its information is not presented to the learner. A more complex constraint-management system to ensure reasonable adherence to rules would give a less severe solution to problems such as this one.

Related to constraint-management is the issue of validating the hypermedia documents. Hypermedia validation is used to determine if the hypermedia application contains the desired information, is built to the desired specifications, and to determine if it has the desired effect upon the user (see Section 2.2.3 for a more complete exploration). Validation of hypermedia is an interesting (and open) research question. APHID presently does none, but validation modules could easily be added to it. As part of the information necessary to provide the ability to customize applications APHID keeps track of most of the data required to do basic structural validation (eg, depth from root, fan-in, fan-out). It could also potentially use information from the concept map to validate the instructional coverage of the application (i.e., correct number of concepts covered and adequate data presented for each concept).

The APHID prototype uses nine types of data elements (novice-critical being an example of one type) and has no facility for ordering elements within each type. This proved to be somewhat problematic, because there was no way of telling the software which of the novice-critical elements should receive priority for selection. A future version of the software should increase the number of types of data elements as well as provide for a more sophisticated selection algorithm that provides for some sort of ordering within each type of element. It may be possible to use the same algorithms that were implemented within the prototype to identify and sort out prerequisite concepts.

In general, the selection method for data elements in the prototype software was simple-minded and must be reviewed. A more sophisticated algorithm based upon more complex relations between data elements is one possibility. This would make the selection of data elements a similar process to the selection of concepts.

The prototype software utilizes two types of traversals and one traversal order to build the different types of hypermedia applications. Even with these two traversal types, several different types of hypermedia are possible. APHID can generate 12 different applications from the algorithms that are presently implemented. A summary of the characteristics of these 12 applications is given in Appendix H. It is possible to imagine how different traversal orders using these same types of traversals would result in new, interesting hypermedia applications.

As new instructional patterns are identified, new traversal techniques must also be incorporated to model those patterns. For instance, it might be possible to use a completely different algorithm (such as a modified hill-climbing algorithm) to effect the traversal of the graph, which would result in a different tutorial-order for the final pages in the application. The choice of algorithm could result in rather disjoint presentations, if it were not carefully orchestrated (consider what a branch and bound approach would do to a presentation), but it is worth investigating further.

In this implementation, XML and XSL are used simply as a convenient method of representing data and applying presentation rules to that data. The technologies related to XML have matured significantly in recent months and could be used for more tasks

within the APHID hypermedia creation process. For instance, all of the presentation patterns for instructional classes could be represented as fragments of an XSL document. The APHID software would then simply select from a list of styles for each type of data element and create a custom style-sheet for each application. Or, APHID could create different style sheets for different portions of the application, allowing, for instance, the remediation portion of an application to be presented differently than the review portion. XSLT can also perform simple selection and sorting, so lists of hyperlinks and elements could be sorted before presentation, if desired.

Data elements for the prototype were stored as large chunks of data. Their XML representation would have allowed APHID to decompose them, and to use small parts of a data element as a part of some other composite data element. The XPath specification (XML Path Language (XPath) 1999) makes possible the precise identification of parts of XML documents, and allows algorithmic selection of those parts. So, future versions of APHID should be able to assemble larger data elements from smaller parts (for instance, it could assemble elements with different titles for different levels of users, or use the same title and attach different explanations to it). This sort of refinement would work well with the improvements for data element selection made earlier. It would require additional guidelines for the composition of elements including well-defined document types, possibly schemas and some additional rules built in to the APHID system.

The APHID software in its present state creates good instructional hypermedia applications. The improvements suggested in this section would make the software more flexible and capable of creating many new types of hypermedia applications. It would also make the software more responsive to the differing needs of users. As the software product becomes more flexible, it can be used in different domains for different purposes. The next section of this chapter explores some possible uses for future versions of APHID.

6.3 Future Directions

While the prototype is targeted at the education and training segment of society, the ideas behind the APHID process are applicable well beyond the domain of instruction. Some of the alternate domains would realize immediate benefits from the software as it is currently implemented; others represent potential applications for future versions of the software. All are equally interesting markets for the APHID style of hypermedia construction.

6.3.1 Industrial Training Materials

One obvious application of the technology is in the domain of industrial training. Industries spend money training and re-training their employees. Some of this money is spent hiring trainers, some is spent creating training materials and much is spent revising materials to bring them up to date or to tailor them for a specific audience. The APHID software would allow trainers to create a set of information for a particular topic and then quickly create training materials for specific audiences as the need arose.

In conjunction with this a tutorial system for the trainers would be helpful. The APHID patterns are useful for communicating ideas to all kinds of people who might be involved in creating training materials. A tutorial or scaffolded environment in which trainers were taught what comprises a good instructional hypermedia system as they constructed one would be quite useful. It would allow trainers who were not educators to learn about what makes good instruction and it would assist instructors to make the transfer between the teaching skills they have and the domain of hypermedia. Such a tutorial system requires better pattern descriptions and a help system that is intertwined with the APHID software.

APHID has the added advantage that maintenance of the assembled information is simplified because all information revisions happen within the APHID software. So, when information changes, or is added to, the changes are made within APHID and new materials are generated from the revised information. There is no need to edit existing

materials because they are simply discarded and replaced with new versions. Creating new versions is simple and painless.

6.3.2 Help Systems

Help systems, especially those that come with modern computer software are often distributed as some form of hypermedia. Authoring help systems for software is a difficult task made worse by the fact that the help pages must be relevant to users with disparate abilities and needs. The different types of users require different levels of help and require different types of help. For instance, some require basic information, some require an indexed reference and some require tutorials. These differing needs are presently partially addressed by providing different methods of accessing the same help pages (indices, keyword searches and hyperlinks for instance). This does allow people to find material in different fashions, but it does nothing to ensure that the material they find is suitably presented for the specific user.

An APHID approach to designing help systems would have all the same advantages for help system construction as it would for the creation of industrial training tools. It would simplify maintenance and ultimately could create help systems that were more useful to a wider audience.

6.3.3 Automated Query Display

In one sense, APHID is a primitive means of automating user-interface creation. It is primitive because it does not generate the actual widgets that comprise the interface, rather it relies on the ability to manipulate an existing, configurable interface framework (in the case of the prototype, the framework is a web browser and the language of manipulation is HTML). There are application domains for which this simple type of interface creation is all that is needed. One example is the domain of information retrieval.

Often information retrieval applications are connected to large federated databases and queries composed in these applications can return thousands of results. Efforts to create

understandable interfaces from such queries have had mixed success (Kleinberg 1998) and generally techniques for sorting and classifying query results are improving.

Under certain, reasonably common, conditions APHID presents another interesting possibility for the management of query results. Suppose that the queries in question are restricted to a specific domain for which there is a large quantity of data. This situation is common in scientific research where large data sets are queried. A search engine returns a large number of hits and the user would like to see something other than a flat-list of those responses. Because the domain is restricted, a concept map of the domain can be created by hand that will be reusable between queries. Each response to the query can be treated as an individual data element and can be assigned to a concept using traditional indexing or keyword approaches to information retrieval (Salton, et al. 1994). Once the concepts in the concept map representing the domain have data elements, APHID can generate pages using any of its available algorithms.

This approach is limited only by the ability to classify query results into data elements. As long as results have enough identifying characteristics to allow them to be classified and assigned to different concepts (and possibly to different element-types) APHID would be capable of creating organized, structured hypermedia out of a flat list of query results. Users would specify the type of organization they desired by selecting a pattern for APHID to follow (in the prototype these patterns are called instructional patterns). Thus users would get query results in a form that matched their own personal preferences.

6.3.4 Adaptive Hypermedia Systems

Probably the most exciting possibility for future work with APHID lies in its applicability to adaptive hypermedia systems. The APHID approach to designing hypermedia has a focus on user preferences, be they end users or instructors. The goal is to present hypermedia applications that comply with the preferences of the user. This goal is congruent with the goals of adaptive hypermedia and, although APHID has some

significant differences from typical adaptive systems, it has complimentary features that would combine well with adaptivity.

Adaptive hypermedia is *" a hypertext or hypermedia system which reflects some features of the user and/or characteristics of his system usage into a user model, and utilizes this model in order to adapt various behavioral aspects of the system to the user" (Brusilovsky, et al. 1998).* In many ways, APHID is similar to an adaptive hypermedia system but there are two key differences. The first is that APHID does not dynamically adapt to a user where most adaptive hypermedia systems perform their adaptations on-the-fly. The second difference is that APHID does not have a model of an individual user, rather it relies on the instructor to supply information about desired characteristics that the final hypermedia should have. APHID does have pre-set values for these characteristics for each of the instructional patterns and user-types, which serve the same function of a user model, but they represent stereotypes rather than an actual individualized model. The assumption made in APHID is that the instructor (or the learner) has a mental model of the intended user and translates that model into values for the attributes of the hypermedia application. So, while on the surface APHID appears similar to adaptive hypermedia systems, it is lacking a refined user model and it creates static hypermedia pages rather than dynamic.

APHID does provide a predictable, configurable mechanism for creating hypermedia given a model of a user (as long as the model can be translated into values for site attributes). APHID could be combined with a system capable of maintaining sophisticated user models, such as (Brusilovsky, et al. 1998) to create a powerful adaptive hypermedia system that responds both to the changing characteristics of users and to the preferences of the site creator.

Because APHID separates the process of creating the presentation from the process of creating the navigation model, it would be a simple task to rework parts of the software to generate pages on-the-fly. The obvious approach to reworking the software is to rewrite APHID so that it creates hypermedia sites dynamically as the user traverses the site. This requires that some part of the software keep track of each user connected to

the system and what their characteristics are. This approach would work, but a simpler approach is possible.

The simple method would be to create a server that caches the XML document that APHID creates for a particular user and sends the appropriate parts of that document to the user when requested. This would allow for automated selection and generation of next and back links and also for the use of a variety of presentation elements, such as color and font, depending upon user history and the changing user model. This approach would not require much alteration to the portion of APHID that supports the instructional designer, nor to the algorithms that create the navigational model. It would require the construction of software to manage the user requests and keep track of the state of the various users, but most user-modelling adaptive systems have such software in place already.

The combination of APHID with techniques for user modelling and adaptivity provide advantages to both approaches. APHID gains the ability to react to specific users rather than simple stereotypes, and also the ability to react dynamically. APHID provides the ability to design the hypermedia based on solid principles and the ability for the designer of the hypermedia application to determine how the application will react to particular user characteristics.

6.4 Conclusions

This research has presented an approach (APHID) to the design of instructional hypermedia applications that is based a set of models describing all aspects of the instructional hypermedia system. APHID uses data models, navigation models and presentation models to represent an instructional hypermedia application. These models are drawn from previous work in hypermedia design and are customized for the APHID approach. The data model consists of a model of instructional classes and a concept map depicting the domain of instruction. The concept map is a familiar construct for many educators, making APHID easy for instructors to work with. APHID augments the navigation model with information about how the instruction is intended to proceed,

which results in a richer navigation model than is usual for most hypermedia development methods. The APHID navigation model addresses specific requirements for instructional hypermedia.

APHID uses patterns to guide the construction of hypermedia applications and to facilitate communication between people involved with that construction. The patterns used describe both interface features and navigational paths through hypermedia applications. These descriptions are paired with an algorithmic representation whenever possible that allows the pattern to be encoded as part of the automated portion of APHID.

The APHID approach is appealing because it addresses deficiencies in present instructional hypermedia development systems. APHID provides support for instructors who may be unfamiliar with the hypermedia system they are using. It provides direct support for making instructional decisions as well as for customizing those decisions to reflect personal preferences. APHID is designed to allow for the addition of new instructional strategies and the development method is independent from the instructional strategy selected. APHID also supports the designer from the initial stages of application development through to the end. The concept mapping environment supports the initial planning and design of instruction as well as the design of the hypermedia application.

APHID is an efficient method to use. Instructors can create initial hypermedia applications in approximately the same amount of time as it takes to create one by hand, but can create subsequent applications, or rapidly revise existing ones. APHID simplifies the maintenance of instructional hypermedia because the data is changed in only one place and the hypermedia applications are simply replaced with new versions. This is not practical for hand-created hypermedia applications. In general, APHID can decrease the amount of time and resources required to create and maintain educationally sound, customized hypermedia applications.

We have shown that APHID can automatically generate acceptable instructional hypermedia applications. Aside from the obvious time savings associated with

automatically generating hypermedia, the fact that it can be automated allows for customization and individualization that is not possible when the hypermedia is created by hand. APHID provides the groundwork for a powerful adaptive hypermedia system that considers both user preferences and instructional preferences when creating applications. Of the numerous possibilities for future work arising from this research, the potential for adaptivity in hypermedia applications is one of the most interesting and should be pursued.

Reference List

Adobe Systems Incorporated. (2000). Adobe Page Mill <http://www.adobe.com/products/pagemill/main.html> (Accessed 23/02/2000).

Anthony, D. L. G. (1995). Patterns for Classroom Education. Proceedings of the Pattern Languages of Programming -PLoP '95 (http://st-www.cs.uiuc.edu/~chai/writing/classroom-ed.html).

Appleton, B. (1997). Patterns and Software: Essential Concepts and Terminology <http://www.enteract.com/~bradapp/docs/patterns-intro.html> (Accessed 28/01/99).

Bernstein, M. (1998). Patterns of Hypertext. Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia- Hypertext 98 (21-29). Pittsburgh.

Bodner, R., Chignell, M., & Tam, T. (1997). Website Authoring using Dynamic Hypertext. Proceedings of WebNet '97 (59-64). Toronto: AACE.

Bos, B. (1999). XML in 10 points <http://www.w3.org/XML/1999/XML-in-10-points> (Accessed 06/02/2000).

Botafogo, R., Rivlin, E., & Shneiderman, B. (1992). Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics. *ACM Transactions on Information Systems, 10* (2), 142-180.

Broder, A., Glassman, S., & Manasse, M. (1997). Syntactic Clustering of the Web. Proceedings of the Sixth International World Wide Web Conference (http://www.scope.gmd.de/info/www6/technical/paper205/paper205.html). Santa Clara, CA.

Brusilovsky, P. (1999). Adaptive and Intelligent Technologies for Web-based Education. In C. Rollinger & C. Peylo (Eds.), *Kunstliche Intelligenz. Vol. 4: Special Issue on Intelligent Systems and Teleteaching* (C. Rollinger & C. Peylo, Eds.) (19-25).

Brusilovsky, P., Kobsa, A., & Vassileva, J. (1998). *Adaptive Hypertext and Hypermedia* (P. Brusilovsky, A. Kobsa & J. Vassileva, Eds.) (v). Kluwer Academic Publishers.

Callan, J., Croft, W. B., & Harding, S. (1992). The INQUERY retrieval system. 3rd International Conference on Database and Expert Systems Applications (78-82).

Chen, P. (1976). The Entity Relationship Model- Towards a Unified view of Data. *ACM Transactions on Database Systems, 1* (1), 9-36.

Chitrenky, K. (1996). Cognitivist Learning Theory <http://www.acs.ucalgary.ca/~kmchitre/cgntv.htm> (Accessed 23/02/2000).

Christoldoulou, S. P., Styliaras, G., & Papatheordorou, T. S. (1998). Evaluation of Hypermedia Application Development and Management Systems. Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia- Hypertext 98 (1-10). Pittsburgh.

Cline, M. P. (1996). The Pros and Cons of Adopting and Applying Design Patterns in the Real World. *Communications of the ACM, 39* (10), 47-49.

Corey, S. M. (1971). The Nature of Instruction. In M. D. Merrill (Ed.), *Instructional Design: Readings* (M. D. Merrill, Ed.) (5-17). Toronto: Prentice-Hall.

Eklund, J. (1995). Cognitive models for structuring hypermedia and implications for learning from the world-wide web. Proceedings of the First Australian World Wide Web Conference- AusWeb95 (http://www.scu.edu.au/sponsored/ausweb/ausweb95/papers/hypertext/eklund/index.html). Ballina.

Firesmith, D., Henderson-Sellers, B., & Graham, I. (1998). *Open Modelling Language (OML) Reference Manual.* Cambridge: Cambridge University Press.

Fowler, M., Scott, K., &. (1997). *UML Distilled : Applying the Standard Object Modeling Language.* Addison-Wesley.

Frohlich, P., & Nejdl, W. (1998). A Database-Oriented Approach to the Design of Educational Hyperbooks. Intelligent Educational Systems on the World Wide Web: Workshop at 8th World Conference of the AIED Society (http://www.contrib.andrew.cmu.edu/~plb/AIED97_workshop/Froehlich/Froehlich.html). Kobe, Japan.

Frohlich, P., Nejdl, W., & Wolpers, M. (1997). KBS-Hyperbook: An Open Hyperbook System for Education. 10th World Conference on Educational Multimedia and Hypermedia (EDMEDIA 98). Freiburg, Germany.

Gagne, R. M., Briggs, L., & Wager, W. (1988). *Principles of Instructional Design.* New York: Holt, Rinehart & Winston.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.

Garrido, A., Rossi, G., & Schwabe, D. (1997). Pattern Systems for Hypermedia. Proceedings of Pattern Languages of Programming-Plop'97 (http://jerry.cs.uiuc.edu/~plop/plop97/Proceedings.ps/garrido.pdf). Allerton, Illinois.

Goldberg, M., & Salari, S. (1997). An Update on WebCT- a Tool for the Creation of Sophisticated Web-Based Learning Environments. Proceedings of NAUWeb '97: Current Practices in Web-Based Course Development (http://about.webct.com/library/full_paper.html). Flagstaff, Arizona.

Harmon, S. W., & Dinsmore, S. (1992). Novice Linking in Hypermedia Environments. Proceedings of the 34th International Conference of the Association for the Development of Computer Based Instructional Systems (http://www.gsu.edu/~wwwitr/docs/novice/index.html). Norfolk VA.

Harold, E. R. (1999). *The XML Bible* (434-444). Foster City: IDG Books Worldwide.

Henderson-Sellers, B. (1998). OPEN <http://www.csse.swin.edu.au/cotar/OPEN/> (Accessed 17/11/98).

Horowitz, E., & Sahni, S. (1978). *Fundamentals of Computer Algorithms* (183-188). USA: Computer Science Press.

IBM Alphaworks. (2000). Technology | overview | xeena <http://www.alphaworks.ibm.com/aw.nsf/techmain/xeena> (Accessed 02/06/2000).

Instructional Management Systems (IMS). (1999). IMS Learning Resource Meta-data <http://www.imsproject.org/metadata/mdinfo01.html> (Accessed 11/04/2000).

Isakowitz, T., Stohr, E. A., & Balasubraminian, P. (1995). RMM: A Methodology for Structured Hypermedia Design. *Communications of the ACM, 38* (8), 34-43.

Jonassen, D. (1991). Objectivism vs Constructivism: Do we need a new philosophical paradigm? *Educational Technology, Research and Development, 39* (3), 5-14.

Joseph, L. (1999). WWW CyberGuide Ratings for Web Site Design <http://www.cyberbee.com/guide2.html>.

Kleinberg, J. M. (1998). Authoritative Sources in a Hyperlinked Environment. Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (668-677). San Franscisco.

Kotlas, C. (1999). Evaluating Web Sites for Educational Uses: Bibliography and Checklist <http://www.unc.edu/cit/guides/irg-49.html> (Accessed 02/07.2000).

Kremer, R. (1997). Constraint Graphs: A Concept Map Meta-Language [Ph.D. Thesis]. Calgary, Alberta: University of Calgary, Department of Computer Science.

Lie, H. W., & Bos, B. (1999). Cascading Style Sheets, level 1 <http://www.w3.org/TR/REC-CSS1> (Accessed 02/07/99).

Manns, M. L. (1998). Pedagogical Patterns: Sample Patterns <http://www-lifia.info.unlp.edu.ar/ppp/samples.htm> (Accessed 02/11/98).

McGrath, J. (1995). Methodology Matters: Doing Research in the Behavioural and Social Sciences. In R. Baecker, J. Grudin, W. Buxton & S. Greenberg (Eds.), *Readings in Human Computer Interaction: Toward the Year 2000* (R. Baecker, J. Grudin, W. Buxton & S. Greenberg, Eds.) (152-169). San Francisco: Morgan Kaufmann.

Merriam-Webster Online. (2000) <http://www.m-w.com/> (Accessed 08/02/2000).

Merrill, M. D. (1996). Instructional Transaction Theory: Instructional Design based on Knowledge Objects. *Educational Technology, 36* (3), 30-37.

Merrill, M. D. (1997). Learning-Oriented Instructional Development Tools. *Performance Improvement, 36* (3), 51-55.

Merrill, M. D. (1998). Knowledge Analysis for Effective Instruction. *CBT solutions*, 1-11.

Meyer, B. (1997). *Object Oriented Software Construction* (2nd) (39-40). Toronto: Prentice Hall.

Mukherjea, S., Foley, J. D., & Hudson, S. (1995). Visualizing Complex hypermedia Networks through Multiple Hierarchical Views. Proceedings of the ACM SIGCHI Conference on Human Factors in Computing, CHI95 (http://www.acm.org/sigchi/chi95/proceedings/papers/sm_bdy.htm). Denver, Colorado.

Murray, T., Condit, C., & Haugsjaa, E. (1998). MetaLinks: A Preliminary Framework for Concept-Based Adaptive Hypermedia. Workshop Proceedings for Workshop on WWW-Based Tutoring- ITS 98. San Antonio.

Murray, T., Condit, C., Piemonte, J., Shen, T., & Kahn, S. (1999). MetaLinks- A Framework and Authoring tool for Adaptive HypermediaS. Lajoie & M. Vivet (Eds.), (S. Lajoie & M. Vivet, Eds.). Proceedings of the 9th International Conference on Artificial Intelligence in Education- AIED 99 (744-746). Le Mans: IOS Press.

Nanard, M., & Nanard, J. (1998). Pushing Reuse in Hypermedia Design: Golden Rules, Design Patterns and Constructive Templates. Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia- Hypertext 98 (11-20). Pittsburgh.

Nielsen, J. (1993). *Usability Engineering*. Boston: Academic Press.

Nielsen, J., & Mack, R. (1994). Usability Inspection Methods: An Executive Summary. In J. Nielsen & R. Mack (Eds.), *Usability Inspection Methods* (J. Nielsen & R. Mack, Eds.) (1-23). John Wiley & Sons.

North, S. (1999). *SAMS Teach Yourself XML in 21 days* (T. Ryan, Ed.) (94-104). Indiana: Sams Net.

Odell, J. (1998). *Advanced Object-Oriented Design Using UML* (91-95). Cambridge: Cambridge University Press.

Oliver, R. (1995). Developing effective hypermedia instructional materials. *Australian Journal of Educational Technology, 11* (2), 8-22.

Paquette, G., & Girard, J. (1996). AGD: A Course Engineering Support System. In C. Frasson, G. Gauthier & A. Lesgold (Eds.), *Intelligent Tutoring Systems: ITS'96* (C. Frasson, G. Gauthier & A. Lesgold, Eds.) (382-391). Berlin: Springer-Verlag.

Paquette, G., Aubin, C., & Crevier, F. (1998). *MISA, A Knowledge-based Method for the Engineering of Learning Systems.*

Pirolli, P., Pitkow, J., & Rao, R. (1996). Silk from a Sow's Ear: Extracting Usable Structures from the Web. Proceedings of the ACM SIGCHI Conference on Human Factors in Computing, CHI96 (http://www.acm.org/sigchi/chi96/proceedings/papers/Pirolli_2/pp2.html). Vancouver.

Pressman, R. (1997). *Software Engineering: A practitioner's approach* (4th) (505-508). New York: McGraw-Hill.

Rational Software Corporation. (1998). UML Modeling Language, Standard Software Notation: Resource Center <http://www.rational.com/uml/> (Accessed 02/11/98).

Reigeluth, C. (1983). Instructional Design: What is it and Why is it? In C. Reigeluth (Ed.), *Instructional-Design Theories and Models: An Overview of their Current Status* (C. Reigeluth, Ed.). Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Rettig, J. (1996). Beyond "Cool"- Analog Models for Reviewing Digital Resources <http://www.onlineinc.com/onlinemag/SeptOL/rettig9.html>.

Ritchie, D. C., & Hoffman, B. (1996). Using Instructional Design Principles To Amplify Learning on the World Wide Web. Proceedings of the Seventh International Conference of the Society for Information Technology and Teacher Education- SITE 96 (http://www.coe.uh.edu/insite/elec_pub/html1996/16teless.htm#ritc). Phoenix, Arizona.

Rivlin, E., Botafogo, R., & Shneiderman, B. (1994). Navigating in Hyperspace: Designing a structure-based toolbox. *Communications of the ACM, 37* (2), 87-96.

Rossi, G., Schwabe, D., & Garrido, A. (1997). Design Reuse in Hypermedia Applications Development. Proceedings of the Eighth ACM Conference on Hypertext and Hypermedia- Hypertext 97 (57-66). Southhampton, Inglaterra: ACM.

Rossi, G., Schwabe, D., Lucena, C., & Cowan, D. (1995). An Object Oriented Model for Designing the Human-Computer Interface of Hypermedia Applications. In *Springer Verlag Workshops in Computing*. International Workshop on Hypermedia Design (IWHD'95) (ftp://ftp.inf.puc-rio.br/pub/docs/techreports/95_07_rossi.ps.gz). Springer Verlag.

Royal Tyrrell Museum. (1997). Royal Tyrrell Museum Virtual Tour <http://tyrrell.magtech.ab.ca/tour/> (Accessed 17/11/98).

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, D., & Lorenzn, W. (1991). *Object-Oriented Modelling and Design*. Prentice-Hall.

Salton, G., Allan, J., & Buckley, C. (1994). Automatic Structuring and Retrieval of Large Text Files. *Communications of the ACM, 37* (2), 97-108.

Saskatchewan Education. (1999). Science Curriculum Guides <http://www.sasked.gov.sk.ca/docs/science.html> (Accessed 02/07/2000).

Schneiderman, B. (1997). *Designing the User Interface:Strategies for Effective Human-Computer Interaction*. Addison-Wesley.

Schwabe, D., Rossi, G., & Barbosa, S. D. J. (1996). Systematic Hypermedia Applications Design with OOHDM. Proceedings of the Seventh ACM Conference on Hypertext and Hypermedia-Hypertext 96 (http://www.cs.unc.edu/~barman/HT96/P52/section1.html). Washington DC.

Seels, B., & Richey, R. (1994). *Instructional Technology: The Definition and Domains of the Field*. Washington, DC: Association for Educational Communications and Technology.

Spertus, E. (1997). ParaSite: Mining Structural Information on the Web. Proceedings of the Sixth International World Wide Web Conference (http://www.scope.gmd.de/info/www6/technical/paper206/paper206.html). Santa Clara, CA.

Spertus, E. (1998). ParaSite: Mining the structural information on the World-Wide Web. [Ph.D. Thesis]. Cambridge, MA: Massachusetts Institute of Technology.

Sun, C.-T., & Ching, Y.-T. (1995). Hypermedia Browsing Pattern Analysis. *International Journal of Educational Telecommunications, 1* (2/3), 293-308.

Tokarchuk, L. (1998). Basic Computer Science Concepts Tutorials <http://www.cs.usask.ca/research/research_groups/aries/projects/applets/tutorials/> (Accessed 07/11/98).

Vassileva, J. (1995). Dynamic Courseware Generation: At the Cross Point of CAL. Proceedings of the International Conference on Computers in Education- ICCE 95 (290-297). Singapore.

Vassileva, J. (1997). Dynamic Courseware Generation on the WWW. Proceedings of the 8th International Conference on Artificial Intelligence and Education (http://julita.usask.ca/homepage/AIED'97.ps). Kobe, Japan.

Wasson (Brecht), B. J. (1990). Determining the Focus of Instruction: Content Planning for intelligent tutoring systems [Diss]. Saskatoon, Canada: Department of Computational Science, University of Saskatchewan.

Wasson, B. (1996). Instructional Planning and Contemporary Theories of Learning: Is this a Self-Contradiction?A. Paiva & J. Self (Eds.), (A. Paiva & J. Self, Eds.). European Conference on Artificial Intelligence in Education (http://www.ifi.uib.no/staff/barbara/papers/Euroaied96.html). Lisbon: Colibri.

Wilson, B. G. (1997). Reflections on Constructivism and Instructional Design. In C. R. Dills & A. A. Romiszowski (Eds.), *Instructional Development Paradigms* (C. R. Dills & A. A. Romiszowski, Eds.). Englewood Cliffs NJ: Educational Technology Publications.

Wilson, B., Jonassen, D., & Cole, P. (1993). Cognitive Approaches to Instructional Design. In G. M. Piskurich (Ed.), *The ASTD handbook of instructional technology* (G. M. Piskurich, Ed.) (21.1-21.22). New York: McGraw-Hill.

Wright, P. (1991). Cognitive overheads and prostheses: Some issues in evaluating hypertexts. Proceedings of the Third ACM Conference on Hypertext- Hypertext 91 (1-12). San Antonio.

XML Path Language (XPath). (1999) <http://www.w3.org/TR/xpath> (Accessed 23/02/2000).

XML Working Group. (1998). Extensible Markup Language (XML) 1.0 <http://www.w3.org/TR/1998/REC-xml-19980210> (Accessed 06/02/2000).

XSL Transformations (XSLT) 1.0. (1999) <http://www.w3.org/TR/xslt> (Accessed 06/02/2000).

Yan, T. W., Jacobsen, M., Garcia-Molina, H., & Dayal, U. (1996). From User Access Patterns to Dynamic Hypertext Linking. Proceedings of the Fifth International World

Wide Web Conference (http://www5conf.inria.fr/fich_html/papers/P8/Overview.html). Paris, France.

Appendix A- Document Type Definitions for Instructional Classes

```
<!ELEMENT description (#PCDATA)>
<!ATTLIST description
 type (intro|summary|conclusion|explanation|preformatted)
"explanation">

<!ELEMENT quiz (description?, quiz_item+)+>
<!ELEMENT quiz_item (description?|(question,answer)+)>
<!ELEMENT question (#PCDATA)>
<!ELEMENT answer (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ATTLIST description
 type (intro|summary|conclusion|explanation|preformatted)
"explanation">

<!ELEMENT question_list (question_item)+>
<!ELEMENT question_item (question, answer?)>
<!ELEMENT question (#PCDATA)>
<!ELEMENT answer (#PCDATA)>

<!ELEMENT narrative (description+)>
<!ELEMENT description (#PCDATA)>

<!ELEMENT list (description+, list_item+)>
<!ELEMENT list_item (#PCDATA)>
<!ELEMENT description (#PCDATA)>

<!ELEMENT instructions (description+, instruction_step+)>
<!ELEMENT instruction_step (description? |action)>
<!ELEMENT description (#PCDATA)>
<!ATTLIST description
 type (intro|summary|conclusion|explanation|preformatted)
"explanation">

<!ELEMENT index (index_entry+)>
<!ELEMENT index_entry (#PCDATA)>

<!ELEMENT FAQ (question,answer)+>
<!ELEMENT question (#PCDATA)>
<!ELEMENT answer (#PCDATA)>

<!ELEMENT example (instance+)>
<!ELEMENT instance (#PCDATA)>
<!ATTLIST instance
 type (positive|negative) "positive"
 media (image|applet|text) "text">

<!ELEMENT vocabulary_list (vocabulary_item+)>
<!ELEMENT vocabulary_item (term, definition)>
<!ELEMENT term (#PCDATA)>
<!ELEMENT definition (#PCDATA)>
```

Appendix B- XML Source for Stacks example

```
<?xml version="1.0"?>
<website>
  <page ref="Stack" title="Stack">
    <links>
      <link type="part">
        <url>Applications.html</url>

        <link_text>Applications</link_text>
      </link>

      <link type="part">
        <url>Representation.html</url>

        <link_text>Representation</link_text>
      </link>

      <link type="part">
        <url>Methods.html</url>

        <link_text>Methods</link_text>
      </link>

      <link type="predecessor">
        <url>AbstractDataTypes.html</url>

        <link_text>Abstract Data Types</link_text>
      </link>
    </links>

    <element_list>
      <data_element>
        <title>Stacks: a description</title>

        <example>
          <instance type="positive" media="text">In order to
          clarify the idea of a stack let's look at a "real life"
          example of a stack. Think of a stack of plates in a high
          school cafeteria. When the plates are being stacked, they
          are added one on top of each other. It doesn't make much
          sense to put each plate on the bottom of the pile, as
          that would be far more work, and would accomplish nothing
          over stacking them on top of each other. Similarily when
          a plate is taken, it is usually taken from the top of the
          stack.</instance>

          <instance type="positive" media="image">
          plates.gif</instance>
        </example>
      </data_element>

      <hr>
      </hr>

      <data_element>
        <title>Stack: a definition</title>

        <vocabulary_list>
          <vocabulary_item>
            <term>Stack</term>
```

```
            <definition>A stack is a homogeneous collection of
            items of any one type, arranged linearly with access at
            one end only, called the top. This means that data can
            be added or removed from only the top. Formally this
            type of stack is called a Last In, First Out (LIFO)
            stack. Data is added to the stack using the Push
            operation, and removed using the Pop
            operation.</definition>
          </vocabulary_item>
        </vocabulary_list>
      </data_element>

      <hr>
      </hr>

      <data_element>
        <title>Stack: the code</title>

        <description type="preformatted">
        <![CDATA[ //-------------------------------------------------------
--------------------------
      #include "stackclass.h"
      //-----------------------------------------------------------------
-------------
      //----Default constructor
      stack::stack(void) {

      Initialize();
      }
      //-----------------------------------------------------------------
-------------
      //----Initialise
      void stack::Initialize(void) {

      //----Set to no items
      NumberOfItems = 0;
      }
      //-----------------------------------------------------------------
-------------
      //----Push
      boolean stack::Push(element Item) {

      //----Check there is space
      if (NumberOfItems < STACK_SIZE) {
      //----Store the item
          TheItems[NumberOfItems] = Item;
          NumberOfItems++;
      //----Return true
          return(TRUE);
          }
      //----If not enough space return false
      else return(FALSE);
      }
      //-----------------------------------------------------------------
-------------
      //----Pop
      boolean stack::Pop(element &Item) {

      //----Check there is an item
      if (!Empty()) {
      //----Take of the item
```

```
            NumberOfItems--;
            Item = TheItems[NumberOfItems];
        //----Return true
            return(TRUE);
            }
        //----If no items return false
        else return(FALSE);
        }
        //-----------------------------------------------------------------
--------------
        //----Empty
        boolean stack::Empty(void) {

        //----Check there are items
        return(NumberOfItems == 0);
        }
        //-----------------------------------------------------------------
--------------
        //----Top of stack
        boolean stack::TopOfStack(element &Item) {

        //---If can pop one, then push it
        if (Pop(Item)) {
            Push(Item);
            return(TRUE);
            }
        else return(FALSE);
        }
        //-----------------------------------------------------------------
--------------
]]>
          </description>
        </data_element>

        <hr>
        </hr>

        <data_element>
          <title>Stack: a simulation</title>

          <simulation type="applet">
            <APPLET CODE="stackGUI.class" width="580" height="430">
              <center>Enable Java to use the applet</center>

              <PARAM name="trays" value="3">
              </PARAM>
            </APPLET>
          </simulation>
        </data_element>

        <hr>
        </hr>
      </element_list>
    </page>

    <page ref="Applications" title="Applications">
      <links>
        <link type="example">
          <url>ArithmeticExpressions.html</url>

          <link_text>Arithmetic Expressions</link_text>
        </link>
```

```
  <link type="example">
    <url>Recursion.html</url>

    <link_text>Recursion</link_text>
  </link>

  <link type="whole">
    <url>Stack.html</url>

    <link_text>Stack</link_text>
  </link>
</links>

<element_list>
  <data_element>
    <title>Examples of stacks</title>

    <list>
      <description>Examples of stacks</description>

      <list_item>a pile of papers</list_item>

      <list_item>Cars in a non-circular driveway</list_item>

      <list_item>Nested function calls</list_item>

      <list_item>Code formatting and expression
      syntax</list_item>

      <list_item>Evaluation of arithmetic
      expressions</list_item>
    </list>
  </data_element>

  <hr>
  </hr>

  <data_element>
    <title>Applications: an intro</title>

    <description type="intro">Stacks are used in many
    applications, including function calls and recursive calls.
    Stacks are also used when processing expressions such as
    arithmetic expressions in polish or reverse polish
    notation.</description>
  </data_element>

  <hr>
  </hr>

  <data_element>
    <title>Applications: some examples</title>

    <list>
      <description type="explanation">There are many uses for
      stacks including:</description>

      <list_item>Providing support for recursive procedure
      calls</list_item>

      <list_item>Searching structures</list_item>
```

```
        <list_item>Computation</list_item>
      </list>
    </data_element>

    <hr>
    </hr>
  </element_list>
</page>

<page ref="ArithmeticExpressions" title="Arithmetic Expressions">
  <links>
    <link type="general">
      <url>Applications.html</url>

      <link_text>Applications</link_text>
    </link>
  </links>

  <element_list>
    <data_element>
      <title>Reverse Polish Notation</title>

      <example>
        <instance media="text">Many people, from mathematicians
        to primary school students, have criticized our standard
        infix representation of expressions as being ambiguous,
        or using a complex system for resolving
        ambiguity.</instance>

        <instance media="text">Consider the expression 2 + 3 * 5.
        Which operation do we do first? Left-to-right makes
        sense, especially to beginners, but we use a complex
        system of priorities to determine which operation is done
        first.</instance>

        <instance media="text">This is also complex to compute,
        as you need to look ahead to determine whether or not it
        is safe to do an operation (hmmm ... how far do you need
        to look ahead). For this reason, alternate notations have
        been developed. One of the easiest is reverse polish
        notation (RPN) in which the operations follows the
        operands. This is also called postfix
        notation.</instance>

        <instance media="text">RPN is traditionally implemented
        using stacks. When you see a number, push it on the
        stack. When you see an operation, pop the operands off
        the stack, do the operation, and push the result back on
        the stack.</instance>

        <instance media="text">For example, to add two to three
        and then multiply by five, we'd use 2 3 + 5 *. Similarly,
        to multiply three and five and then add 2, we might write
        2 3 5 * +</instance>

        <instance media="text">RPN is unambiguous and requires no
        parenthesization. It is used by a number of calculators
        (particularly HP's) and is supported by the Unix program
        dc.</instance>
      </example>
    </data_element>
```

```
    <hr>
    </hr>
  </element_list>
</page>

<page ref="Recursion" title="Recursion">
  <links>
    <link type="general">
      <url>Applications.html</url>

      <link_text>Applications</link_text>
    </link>

    <link type="predecessor">
      <url>LinkedList.html</url>

      <link_text>Linked List</link_text>
    </link>
  </links>

  <element_list>
    <data_element>
      <title>Recursion</title>

      <description>Recursion is nothing more than a function that
      calls itself. It is therefore in a loop which must have a
      way of terminating. When you called the function from
      itself, it stored all of the variables and all of the
      internal flags it needs to complete the function in a block
      somewhere. The next time it called itself, it did the same
      thing, creating and storing another block of everything it
      needed to complete that function call. It continued making
      these blocks and storing them away until it reached the
      last function when it started retrieving the blocks of
      data, and using them to complete each function call. The
      blocks were stored on an internal part of the computer
      called the "stack". This is a part of memory carefully
      organized to store data just as described above. A stack is
      used in nearly all modern computers for internal
      housekeeping chores.</description>
    </data_element>

    <hr>
    </hr>
  </element_list>
</page>

<page ref="LinkedList" title="Linked List">
  <links>
    <link type="general">
      <url>Representation.html</url>

      <link_text>Representation</link_text>
    </link>

    <link type="successor">
      <url>Recursion.html</url>

      <link_text>Recursion</link_text>
    </link>
  </links>
```

```
  <element_list>
    <data_element>
      <title>Linked Lists</title>

      <description type="intro">To allow for a stack with a
      variable maximum size, it can be implemented as a linked
      list using pointers . When a stack is implemented this way,
      each node contains a data field for the information, and a
      pointer to the next node on the list. Top is a pointer to
      the top of the list. When top is NULL (or NIL, depending on
      the programming language), the stack is
      empty.</description>
    </data_element>

    <hr>
    </hr>

    <data_element>
      <title>More on Linked Lists</title>

      <description type="explanation">Implementing stacks as
      linked lists provides a solution to the problem of
      dynamically growing stacks, as a linked list is a dynamic
      data structure. The stack can grow or shrink as the program
      demands it to. However, if a small and/or fixed amount of
      data is being dealt with, it is often simpler to implement
      the stack as an array.</description>
    </data_element>

    <hr>
    </hr>
  </element_list>
</page>

<page ref="Representation" title="Representation">
  <links>
    <link type="example">
      <url>LinkedList.html</url>

      <link_text>Linked List</link_text>
    </link>

    <link type="example">
      <url>Array.html</url>

      <link_text>Array</link_text>
    </link>

    <link type="whole">
      <url>Stack.html</url>

      <link_text>Stack</link_text>
    </link>
  </links>

  <element_list>
    <data_element>
      <title>Representation using Arrays</title>

      <description type="explanation">Array-based (or
      vector-based) implementations are fairly easy. We simply
```

```
      add elements to the end of the array, and remove elements
      from the end of the array. We may need to keep track of the
      position of the last element, but we can also use the
      size() or length to determine that.</description>
    </data_element>

    <hr>
    </hr>

    <data_element>
      <title>Representing Stacks</title>

      <description type="explanation">There are multiple
      implementations of stacks. The implementation we choose may
      depend on our knowledge of the use of the stack (e.g., are
      we going to have a limited or unlimited size?) and the
      limitations of our programming language (e.g., does it
      support dynamic data allocation). The two most basic ways
      to implement stacks are with vectors and
      lists.</description>
    </data_element>

    <hr>
    </hr>

    <data_element>
      <title>Representation using lists</title>

      <description type="explanation">A list-based implementation
      is relatively straightforward, as many list implementations
      include all the standard stack operations. push(o) is just
      another name for insertAtFront(o) pop() is just another
      name for removeFromFront() peek() is just another name for
      head()</description>
    </data_element>

    <hr>
    </hr>
  </element_list>
</page>

<page ref="Array" title="Array">
  <links>
    <link type="general">
      <url>Representation.html</url>

      <link_text>Representation</link_text>
    </link>
  </links>

  <element_list>
    <data_element>
      <title>Arrays: an introduction</title>

      <description type="intro">The purpose of an array is to
      store information in an organized way. It allows the
      programmer to group data together. It provides a convenient
      way for handling large amounts of data, due to the fact
      that the arrays indexing structure is easily manipulated
      with iterative control structures. The main disadvantage to
      using an array is that it is a static data structure. The
      array must be formally declared before using it. This
```

```
  involves knowing the amount of data that will be written
  into the array beforehand. This often leads to the
  programmer "guessing high" when deciding upon array bounds,
  so that there will be enough room available in the array.
  This can be very inefficient memory wise.</description>
</data_element>

<hr>
</hr>

<data_element>
  <title>Stacks as arrays</title>

  <description type="intro">One of two ways to implement a
  stack is by using a one dimensional array (also known as a
  vector). When implemented this way, the data is simply
  stored in the array. Top is an integer value, which
  contains the array index for the top of the stack. Each
  time data is added or removed, top is incremented or
  decremented accordingly, to keep track of the current top
  of the stack. By convention, an empty stack is indicated by
  setting top to be equal to -1.</description>
</data_element>

<hr>
</hr>

<data_element>
  <title>Array Representation</title>

  <description type="explanation">Stacks implemented as
  arrays are useful if a fixed amount of data is to be used.
  However, if the amount of data is not a fixed size or the
  amount of the data fluctuates widely during the stack's
  life time, then an array is a poor choice for implementing
  a stack.</description>
</data_element>

<hr>
</hr>

<data_element>
  <title>Array: an example</title>

  <example>
    <instance type="negative" media="text">For example,
    consider a call stack for a recursive procedure. It can
    be difficult to know how many times a recursive procedure
    will be called, making it difficult to decide how large
    the maximum stack size (array size) should be.
    Additionally, the recursive procedure may not be called
    the same number of times each time it is invoked- meaning
    that the stack size changes at run time. An array
    representation for the stack would be a poor choice, as
    you would have to declare it to be large enough that
    there is no danger of it running out of storage space
    when the procedure recurses many times. This would waste
    a significant amount of memory if the procedure normally
    only recurses a few times.</instance>
  </example>
</data_element>
```

```
    <hr>
    </hr>
  </element_list>
</page>

<page ref="Methods" title="Methods">
  <links>
    <link type="example">
      <url>Push.html</url>

      <link_text>Push</link_text>
    </link>

    <link type="example">
      <url>Pop.html</url>

      <link_text>Pop</link_text>
    </link>

    <link type="example">
      <url>Top.html</url>

      <link_text>Top</link_text>
    </link>

    <link type="whole">
      <url>Stack.html</url>

      <link_text>Stack</link_text>
    </link>
  </links>

  <element_list>
    <data_element>
      <title>Methods: an introduction</title>

      <vocabulary_list>
        <vocabulary_item>
          <term>Push</term>

          <definition>Adds or pushes another item onto the stack.
          The number of items on the stack must be less than the
          stack maximum.</definition>
        </vocabulary_item>

        <vocabulary_item>
          <term>Pop</term>

          <definition>This operation removes an item from the
          stack and returns the item. The number of items on the
          stack must be greater than 0.</definition>
        </vocabulary_item>

        <vocabulary_item>
          <term>Top</term>

          <definition>This operation returns the value of the
          item at the top of the stack. The operation does not
          change the stack.</definition>
        </vocabulary_item>

        <vocabulary_item>
```

```
          <term>Is Empty</term>

          <definition>This boolean operation returns true if the
          stack is empty and false if it is not.</definition>
        </vocabulary_item>

        <vocabulary_item>
          <term>Is Full</term>

          <definition>This operation returns true if the stack is
          full and false if it is not.</definition>
        </vocabulary_item>
      </vocabulary_list>
    </data_element>

    <hr>
    </hr>

    <data_element>
      <title>Methods: a review</title>

      <example>
        <instance type="positive" media="image">
        stack_arr.gif</instance>
      </example>
    </data_element>

    <hr>
    </hr>
  </element_list>
</page>

<page ref="Push" title="Push">
  <links>
    <link type="general">
      <url>Methods.html</url>

      <link_text>Methods</link_text>
    </link>
  </links>

  <element_list>
    <data_element>
      <title>Push: the algorithm</title>

      <instructions>
        <description>In order to understand how the Push function
        operates, we need to look at the algorithm in more
        detail.</description>

        <description type="preformatted">procedure Push(item :
        items); //add item to the stack //top is the current top
        of stack //n is the maximum stack size begin if top = n
        then stack full; top := top+1; stack(top) := item; end:
        //of Push</description>

        <instruction_step>
          <action>procedure Push(item : items);</action>

          <description>Push requires a parameter - item. This
          parameter is of the same data type as the rest of the
          stack. Item is the data to be added to the
```

```
        stack.</description>
      </instruction_step>

      <instruction_step>
        <action>if top = n then stack full;</action>

        <description>This line performs a check to see whether
        or not the stack is full. If it is, then some error
        condition, such as an error handling routine, is
        triggered. If it is not,the procedure continues. If the
        stack is implemented as linked list, you simply can not
        compare top to n to see if the stack is full. Top is a
        pointer, and will not be equal n when the stack is
        full. The easiest solution to this problem is to keep a
        counter, and increment or decrement it accordingly each
        time you push or pop. This counter will contain the
        number of items on the stack, and can be used for
        condition testing.</description>
      </instruction_step>

      <instruction_step>
        <action>top := top+1;</action>

        <description>If the stack is not full, top is increased
        by a value equal to the size of another item. This
        allocates room for the insertion. If implementing the
        stack as a linked list, allocation must be handled
        differently.Allocation of space for the item to be
        inserted is dependent on the language you are using.
        What you need to do is create/declare/initialize a new
        pointer, which can be used to reference the new node.
        This node must then be linked to the stack,by
        referencing it from the previous node.</description>
      </instruction_step>

      <instruction_step>
        <action>stack(top) := item;</action>

        <description>After the room for another item has been
        added, the data is then inserted. This is done by
        setting the new value of top to be equal to the item to
        be inserted.</description>
      </instruction_step>
    </instructions>
  </data_element>

  <hr>
  </hr>
 </element_list>
</page>

<page ref="Pop" title="Pop">
 <links>
  <link type="general">
   <url>Methods.html</url>

   <link_text>Methods</link_text>
  </link>
 </links>

 <element_list>
  <data_element>
```

```
      <title>Pop: the algorithm</title>

      <instructions>
        <description>Data is removed from a stack by using Pop.
        From a procedural perspective, pop is called with the
        line Pop(item), where item is the variable to store the
        popped item in. Pop returns the item at the top of the
        stack (and only that item).</description>

        <description type="preformatted">procedure Pop(var item :
        items); //remove top element from the stack and put it in
        the item begin if top = -1 then stackempty; item :=
        stack(top); top := top-1; end; //of Pop</description>

        <instruction_step>
          <action>procedure Pop(var item : items);</action>

          <description>Pop is called with the parameter item.
          This variable is where the item popped off the top of
          the stack is stored for later access.</description>
        </instruction_step>

        <instruction_step>
          <action>if top = -1 then stackempty;</action>

          <description>If top is equal to -1, the stack is empty
          - there is no item to pop off the stack. Control can
          then be passed to an error handling routine. If the
          stack is implemented as a linked list, the stack will
          be empty if top = NULL/NIL.</description>
        </instruction_step>

        <instruction_step>
          <action>item := stack(top);</action>

          <description>Set item to be equal to the data in the
          top node.</description>
        </instruction_step>

        <instruction_step>
          <action>top := top-1;</action>

          <description>This statement removes the top item from
          the stack. Decrement top by 1 so that it now accesses
          the new top of the stack. If implementing the stack as
          a linked list, this step is done by setting top to
          point to the next item on the stack, which will become
          the current top. In addition, if implementing the stack
          as a linked list, it is necessary to add a statement
          that will decrement the count variable that keeps track
          of the number of items on the stack.</description>
        </instruction_step>
      </instructions>
    </data_element>

    <hr>
    </hr>
  </element_list>
</page>

<page ref="Top" title="Top">
  <links>
```

```
  <link type="general">
    <url>Methods.html</url>

    <link_text>Methods</link_text>
  </link>
</links>

<element_list>
  <data_element>
    <title>Top: the algorithm</title>

    <instructions>
      <description type="preformatted">procedure Top(var item :
      items); //return top element from the stack stack and put
      it in the item begin if top = -1 then stackempty; item :=
      stack(top); end; //of Top</description>

      <instruction_step>
        <action>procedure Pop(var item : items);</action>

        <description>Pop is called with the parameter item,
        which is the variable in which the popped item is to be
        stored.</description>
      </instruction_step>

      <instruction_step>
        <action>if top = -1 then stackempty;</action>

        <description>If top is equal to -1, the stack is empty.
        Control can then be passed to an error handling
        routine. If the stack is implemented as a linked list,
        the stack will be empty if top =
        NULL/NIL.</description>
      </instruction_step>

      <instruction_step>
        <action>item := stack(top);</action>

        <description>Set item to be equal to the data in the
        top node.&gt;</description>
      </instruction_step>
    </instructions>
  </data_element>

  <hr>
  </hr>

  <data_element>
    <title>Top: a description</title>

    <description type="intro">What if you want to access the
    data field at the top of the stack, but you do not want to
    remove the item? It seems like a hassle (not to mention
    that it is inefficient) to pop the top item, use it, and
    push it back onto the stack. The solution to this problem
    is the function Top (do not confuse this function with the
    variable top). Top will return the data segment of the top
    item without removing it from the stack. It is called with
    Top(item), where item is a variable that will hold the data
    item.</description>
  </data_element>
```

```
      <hr>
      </hr>
    </element_list>
  </page>

  <page ref="AbstractDataTypes" title="Abstract Data Types">
    <links>
      <link type="successor">
        <url>Stack.html</url>

        <link_text>Stack</link_text>
      </link>
    </links>

    <element_list>
      <data_element>
        <title>ADT: a description</title>

        <description type="explanation">An Abstract Data Type (ADT)
        is a set of allowed data values, with a set of allowed
        operations. This allows the user to create data types to
        their own specifications. All data must conform to the
        predefined structure, and this data may only be manipulated
        through a set of predefined operations. These operations
        must not violate the integrity of the data structure. The
        purpose of the ADT is to shield the programmer from the
        internal workings of the data structure, by giving access
        to the data only through the operations.</description>
      </data_element>

      <hr>
      </hr>
    </element_list>
  </page>
</website>
```

Appendix C- XSL Style Sheets

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
xmlns:xt="http://www.jclark.com/xt"
extension-element-prefixes="xt">
  <xsl:strip-space elements="description * " />

  <xsl:variable name="DEBUG">
  </xsl:variable>

  <xsl:variable name="DETAILS">1</xsl:variable>

<!-- ******************************************************* -->
<!--template for web site      -->
<!--                    <!ELEMENT website (page+)>-->
<!-- ******************************************************* -->
  <xsl:template match="website">
    <xsl:apply-templates select="page" />
  </xsl:template>

<!-- ******************************************************* -->
<!--template for pages -->
<!--    ELEMENT page (links, element_list)  -->
<!-- ATTLIST page
    ref ID #REQUIRED
    title CDATA #REQUIRED>                                    -->
<!-- ******************************************************* -->
  <xsl:template match="page">
    <xt:document href="{concat(@ref,'.html')}">
      <html>
        <head>
          <title>
            <xsl:value-of select="@title" />
          </title>

          <LINK rel="stylesheet" type="text/css"
          href="../geostyle.css" title="style1" />

<style>

b A:link {color:#990099; font-size: 8pt}
<!--next links-->

i A:link {color:#000088; font-size: 8pt}
<!--back links -->

em A:link  {color: #996600; font-size: 8pt}
<!--down links-->

A:link  {font-size: 8pt}
cite {font-size: 6pt}

</style>
        </head>

        <body>
          <A NAME="top">
          </A>

          <center>
            <h1>
```

```
            <xsl:value-of select="@title" />
          </h1>
        </center>

        <table cellpadding="5">
          <tr>
            <td VALIGN="Top">
              <xsl:apply-templates select="links" />
            </td>

            <td>
              <xsl:apply-templates select="element_list" />
            </td>
          </tr>
        </table>

        <CENTER>
          <A HREF="#top">top</A>
        </CENTER>
      </body>
    </html>
  </xt:document>
</xsl:template>

<!-- ******************************************************** -->
<!--templates for links                                       -->
<!-- <!ELEMENT links (link+)>-->
<!--ELEMENT link (link_text, url)-->
<!--PART,EXAMPLE,SUCCESSOR,WHOLE,GENERAL,PREDECESSOR,PARENT,
CHILD,PRESIB,POSTSIB,NEXT,BACK-->
<!-- ******************************************************** -->
<xsl:template match="links">
  <xsl:apply-templates select="link" />
</xsl:template>

<!--links that go down the hierarchy-->
<xsl:template

match="link[@type='PART']|link[@type='EXAMPLE']|link[@type='SUCCESSOR']
|link[@type='CHILD']|link[@type='POSTSIB']">

  <xsl:if test="$DEBUG=1">
    <xsl:value-of select="@type" />
  </xsl:if>

  <em>
    <a href="{normalize(url)}">
      <xsl:value-of select="link_text" />
    </a>
  </em>

  <cite>--Detailed Topic--</cite>

  <br>
  </br>
</xsl:template>

<!--links that climb up the hierarchy-->
<xsl:template

match="link[@type='WHOLE']|link[@type='GENERAL']|link[@type='PREDECESSO
R']|link[@type='PRESIB']|link[@type='PARENT']">
```

```
    <xsl:if test="$DEBUG=1">
      <xsl:value-of select="@type" />
    </xsl:if>

    <a href="{normalize(url)}">
      <xsl:value-of select="link_text" />
    </a>

    <cite>--General Topic--</cite>

    <br>
    </br>
  </xsl:template>

<!--parent link-->
  <xsl:template match="link[@type='PARENT']">
    <xsl:if test="$DEBUG=1">
      <xsl:value-of select="@type" />
    </xsl:if>

    <i>_______</i>

    <br>
    </br>

    <cite>--Same Topic-easier--</cite>

    <br>
    </br>

    <a href="{normalize(url)}">
      <xsl:value-of select="link_text" />
    </a>
  </xsl:template>

  <xsl:template match="link[@type='HOME']">
    <xsl:if test="$DEBUG=1">
      <xsl:value-of select="@type" />
    </xsl:if>

    <a href="{normalize(url)}">
      <FONT COLOR="#FF0099">
        <i>Back to Beginning</i>
      </FONT>
    </a>

    <br>
    </br>
  </xsl:template>

  <xsl:template match="link[@type='NEXT']">
    <xsl:if test="$DEBUG=1">
      <xsl:value-of select="@type" />
    </xsl:if>

    <b>
      <a href="{normalize(url)}">
        <xsl:value-of select="link_text" />
      </a>
    </b>
```

```
    <cite>--Next--</cite>

    <br>
    </br>
  </xsl:template>

  <xsl:template match="link[@type='BACK']">
    <xsl:if test="$DEBUG=1">
      <xsl:value-of select="@type" />
    </xsl:if>

    <i>
      <a href="{normalize(url)}">
        <xsl:value-of select="link_text" />
      </a>
    </i>

    <cite>--Back--</cite>

    <br>
    </br>

    <i>_______</i>

    <br>
    </br>
  </xsl:template>

  <xsl:template match="link">
    <xsl:if test="$DEBUG=1">
      <xsl:value-of select="@type" />
    </xsl:if>

    <a href="{normalize(url)}">
      <xsl:value-of select="link_text" />
    </a>

    <br>
    </br>
  </xsl:template>

  <xsl:template match="url">
    <xsl:value-of select="." />
  </xsl:template>

  <xsl:template match="link_text">
    <xsl:value-of select="." />
  </xsl:template>

<!-- ******************************************************* -->
<!--templates for data-elements                              -->
<!--<!ELEMENT element_list (data_element+)> -->
<!-- <!ELEMENT data_element (title?,
description?,(question_list|example|instructions|index|simulation|pract
ise_problem|quiz|FAQ|description))>
-->
<!--<!ATTLIST data_element
id ID #REQUIRED
name CDATA #IMPLIED
importance (critical | explain | enrich) "explain"
level (beginner | intermediate | advanced) "beginner" -->
<!-- ******************************************************* -->
```

```
  <xsl:template match="element_list">
    <xsl:apply-templates select="data_element" />
  </xsl:template>

  <xsl:template match="data_element">
    <xsl:apply-templates />
  </xsl:template>

  <xsl:template match="data_element/title">
    <h3>
      <xsl:value-of select="." />
    </h3>
  </xsl:template>

<!-- ******************************************************** -->
<!--List-->
<!--ELEMENT list (description+,list_item+)-->
<!-- ******************************************************** -->
  <xsl:template match="list">
    <xsl:for-each select="description">
      <p>
        <xsl:value-of select="." />
      </p>
    </xsl:for-each>

    <ul>
      <xsl:for-each select="list_item">
        <li>
          <xsl:value-of select="." />
        </li>
      </xsl:for-each>
    </ul>
  </xsl:template>

<!-- ******************************************************** -->
<!--Question_list-->
<!--ELEMENT question_list (question_item)+-->
<!-- ******************************************************** -->
  <xsl:template match="question_list">
    <ol>
      <xsl:for-each select="question_item">
        <li>
          <xsl:apply-templates />
        </li>
      </xsl:for-each>
    </ol>
  </xsl:template>

  <xsl:template match="question_item">
    <li>
      <xsl:apply-templates select="answer" />

      <dir>
        <i>
          <xsl:value-of select="answer" />
        </i>
      </dir>
    </li>
  </xsl:template>

  <xsl:template match="question">
    <b>
```

```
      <xsl:value-of select="." />
    </b>
  </xsl:template>

  <xsl:template match="answer">
    <dir>
      <i>
        <xsl:value-of select="." />
      </i>
    </dir>
  </xsl:template>

<!-- ******************************************************* -->
<!--Example-->
<!--ELEMENT example (instance+)-->
<!-- Instance-->
<!--ELEMENT instance (#PCDATA)-->
<!--ATTLIST instance
 type (positive|negative) "positive"
    media (image|applet|text|preformatted) "text"-->
<!-- ******************************************************* -->
  <xsl:template match="example">
    <xsl:apply-templates select="instance" />
  </xsl:template>

  <xsl:template match="instance">
    <xsl:choose>
      <xsl:when test="@media='preformatted'">
<pre>

<xsl:value-of select="." />


</pre>
      </xsl:when>

      <xsl:when test="@media='applet'">
        <center>
          <APPLET CODE="{.}" width="580" height="430">
          </APPLET>
        </center>
      </xsl:when>

      <xsl:when test="@media='image'">
        <img src="{.}" />

        <br>
        </br>
      </xsl:when>

      <xsl:otherwise>
        <p>
          <xsl:value-of select="." />
        </p>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

<!-- ******************************************************* -->
<!--Instructions-->
<!--ELEMENT instructions (description?, instruction_step+)-->
<!--ELEMENT instruction_step (description? |action)-->
```

```
<!-- ********************************************************* -->
  <xsl:template match="instructions">
    <xsl:apply-templates select="description" />

    <ol>
      <xsl:for-each select="instruction_step">
        <li>
          <xsl:apply-templates select="action" />
        </li>

        <dir>
          <i>
            <xsl:apply-templates select="description" />
          </i>
        </dir>
      </xsl:for-each>
    </ol>
  </xsl:template>

  <xsl:template match="instruction_step">
    <li>
      <xsl:apply-templates select="action" />
    </li>

    <dir>
      <i>
        <xsl:apply-templates select="description" />
      </i>
    </dir>
  </xsl:template>

  <xsl:template match="action">
    <xsl:value-of select="." />
  </xsl:template>

<!-- ********************************************************* -->
<!--Index-->
<!--ELEMENT index (index_entry+)-->
<!-- ********************************************************* -->
  <xsl:template match="index">
    <xsl:apply-templates />
  </xsl:template>

  <xsl:template match="index_entry">
    <p>
      <xsl:value-of select="." />
    </p>
  </xsl:template>

<!-- ********************************************************* -->
<!--FAQ-->
<!--ELEMENT FAQ (question,answer)+-->
<!-- ********************************************************* -->
  <xsl:template match="FAQ">
    <xsl:apply-templates />
  </xsl:template>

<!-- ********************************************************* -->
<!--Vocab List-->
<!--ELEMENT vocabulary_list (vocabulary_item+)-->
<!--ELEMENT vocabulary_item (term, definition)-->
<!--ELEMENT term (#PCDATA)-->
```

```
<!--ELEMENT definition (#PCDATA)-->
<!-- ********************************************************* -->
  <xsl:template match="vocabulary_list">
    <table>
      <xsl:for-each select="vocabulary_item">
        <tr>
          <td>
            <xsl:apply-templates select="term" />
          </td>

          <td>
            <xsl:apply-templates select="definition" />
          </td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:template>

  <xsl:template match="vocabulary_item">
    <xsl:apply-templates />
  </xsl:template>

  <xsl:template match="term">
    <b>
      <xsl:value-of select="." />
    </b>
  </xsl:template>

  <xsl:template match="definition">
    <i>
      <xsl:value-of select="." />
    </i>
  </xsl:template>

<!-- ********************************************************* -->
<!--Quiz-->
<!--ELEMENT quiz (description?, quiz_item+)+-->
<!--ELEMENT quiz_item (description?|(question,answer)+)-->
<!--ELEMENT question (#PCDATA)-->
<!--ELEMENT answer (#PCDATA)-->
<!-- ********************************************************* -->
  <xsl:template match="quiz">
    <xsl:apply-templates />
  </xsl:template>

  <xsl:template match="quiz_item">
    <xsl:apply-templates />
  </xsl:template>

<!-- ********************************************************* -->
<!--Narrative-->
<!--ELEMENT narrative (description+)-->
<!-- ********************************************************* -->
  <xsl:template match="narrative">
    <xsl:apply-templates match="description" />
  </xsl:template>

<!-- ********************************************************* -->
<!--Description-->
<!--ELEMENT description (#PCDATA)-->
<!--ATTLIST description
```

```
 type (intro|summary|conclusion|explanation|preformatted)
"explanation"-->
<!-- ******************************************************* -->
  <xsl:template match="description">
    <xsl:choose>
      <xsl:when test="@type='preformatted'">
<pre>

<xsl:value-of select="." />


</pre>
      </xsl:when>

      <xsl:otherwise>
        <p>
          <xsl:value-of select="." />
        </p>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

<!-- ******************************************************* -->
<!--simulation-->
<!--ELEMENT simulation (#PCDATA)-->
<!--may want to make this a CDATA-->
<!--ATTLIST simulation
media  (image|applet|text) "applet"-->
<!-- ******************************************************* -->
  <xsl:template match="simulation">
    <xsl:choose>
      <xsl:when test="@media='text'">
        <p>
          <xsl:value-of select="." />
        </p>
      </xsl:when>

      <xsl:when test="@media='applet'">
        <center>
          <xsl:value-of disable-output-escaping="yes" select="." />
        </center>
      </xsl:when>

      <xsl:when test="@media='image'">
        <img src="{normalize(.)}" />

        <br>
        </br>
      </xsl:when>
    </xsl:choose>
  </xsl:template>

<!-- ******************************************************* -->
<!--practise problem-->
<!--ELEMENT practise_problem (#PCDATA)-->
<!--ATTLIST practise_problem
media (image|applet|text) "text"-->
<!-- ******************************************************* -->
  <xsl:template match="practise_problem">
    <xsl:choose>
      <xsl:when test="@media='text'">
        <p>
```

```
          <xsl:value-of select="." />
        </p>
      </xsl:when>

      <xsl:when test="@media='applet'">
        <center>
          <APPLET CODE="{.}" width="580" height="430">
          </APPLET>
        </center>
      </xsl:when>

      <xsl:when test="@media='image'">
        <img src="{.}" />

        <br>
        </br>
      </xsl:when>
    </xsl:choose>
  </xsl:template>

<!-- ******************************************************* -->
<!--templates for PCDATA elements -->
<!-- ******************************************************* -->
  <xsl:template match="title">
    <b>
      <xsl:value-of select="." />
    </b>
  </xsl:template>

  <xsl:template match="summary">
    <xsl:value-of select="." />
  </xsl:template>
</xsl:stylesheet>
```

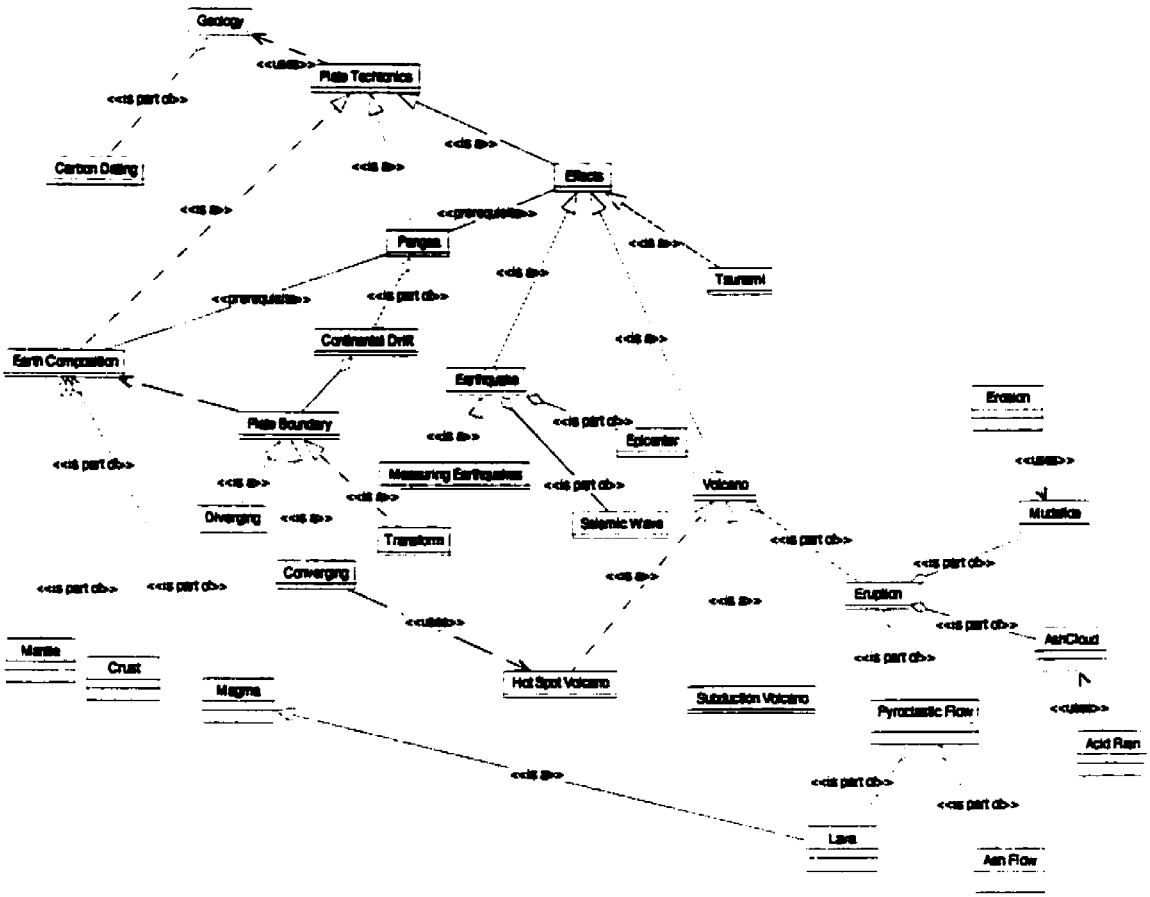
Appendix D- Instructions to Study Participants

You are asked to build two different web sites from the materials on geology and plate tectonics. The web site s may contain the any combination of materials you choose (re-use things between the sites if you like). You do not have to use all of the materials provided, but you may not add additional material (even cut/paste titles please).

Your web sites will be compared in a blind study to other web sites. To make the comparison fair, we ask that you use the style sheet that we provide and that you are consistent about placing links only on the 'link' portion of the pages. (I recognize that this may not be the 'best' design.. but it is the easiest to make consistent).

One of your sites should present the material as it would be presented to students who are beginning their study of Plate Tectonics. These students would be unfamiliar with even the basic concepts concerning Plate Tectonics. The students are relatively immature (grade 7/8).

Your other site should present the material as a review to a more mature student who requires remedial teaching in the concept of Earthquakes. This student has been taught the material previously but has demonstrated difficulty with it, and is especially lacking an understanding of Earthquakes.



Appendix E- Comments from Participants

	A	B	C
What types of activities consumed the bulk of your time?	Sorting and organizing the material.	Cutting/pasting	Sorting and organizing the material.
Was it easier or more difficult to construct your second site?	Easier	Easier	easier
What was the most difficult (or dreary) aspect of the website construction?	Fitting the material together. Restriction on creating new content.	Restriction on creating new content.	Fitting the material together.
Which parts of the construction process do you feel could be automated?	Navigational tools and format could be automated. Content selection done by a human	Indexing of content	Template files Indexing content
Do you feel that a website targeted towards a specific type of user would be a valuable educational tool?	yes	yes	yes
Would you be willing to construct a third type of site from the same set of materials?	yes	More info first	yes
Additional comments		Need a complete topic tree and more instructional structure.	Restriction on creating new content and editing was troublesome.

Appendix F- Instructions to Evaluator

Background

Instructional hypermedia materials (namely web sites) are usually constructed to be general-purpose sites due to time constraints and designer inexperience. We feel that better instructional sites can be built if a specific instructional purpose is identified before building the site. Although free exploration has a prominent place in learning and in hypermedia, so does direct instruction. Hypermedia applications can be constructed to provide some of the features of direct instruction such as ordering and spiraled presentation of concepts.

The sites to be evaluated have each been constructed (by different people) with a specific instructional goal in mind. Each site-builder used a different approach to the design of the site. Each approach may have strengths and weaknesses. This study is intended to get an assessment of those strengths and weaknesses as they are illustrated in the finished web sites.

This study is concerned with **organisation** and **structure** of hypermedia only. All the sites have been normalized with the same font, layout and colour scheme. The layout chosen has all the information on the right hand side of the screen, and all the hyperlinks on the left-hand side. While this is likely not an optimal layout for interesting hypermedia, it is one of the easiest to calculate measures of site size and complexity from and is adequate for the purposes of this study. To further normalize the sites, each was constructed using the same bank of data, although no one site presents all of the available data. To ensure consistency of data across sites, and to improve the longevity of the sites, only hyperlinks to pages within the web site were permitted (no off-site links).

Format of the Evaluation

Ten different instructional web sites have been developed. Five were designed to present material to novice learners, five were designed to present the same concepts to advanced learners who required remediation. (See the original task description for more information).

Complete the attached Website Evaluation Sheet for each site (10 sheets in total). All the criteria should be considered from the perspective of the target audience. For example, when rating "navigability", rate whether the navigability is suitable for the target audience. Feel free to include written comments as well. Before rating the sites, please read the criterion descriptions.

The Web Site Creation Tasks

Below is the task description given to the participants who created the web sites for this study.

> *You are asked to build two different web sites from the materials on geology and plate tectonics. The web sites may contain the any combination of materials you choose (re-use things between the sites if you like). You do not have to use all of the materials provided, but you may not add additional material (even cut/paste titles please).*
>
> *Your web sites will be compared in a blind study to other web sites. To make the comparison fair, we ask that you use the style sheet that we provide and that you are consistent about placing links only on the 'link' portion of the pages. (I recognize that this may not be the 'best' design.. but it is the easiest to make consistent).*
>
> *One of your sites should present the material as it would be presented to students who are beginning their study of Plate Tectonics. These students would be unfamiliar with even the basic concepts concerning Plate Tectonics. The students are relatively immature (grade 7/8).*
>
> *Your second site should present the material as a review for a more mature student who requires remedial teaching in the concept of Earthquakes. This student has been taught the material previously but has demonstrated difficulty with it, and is especially lacking an understanding of Earthquakes.*

Descriptions of Rating Criteria

These descriptions indicate the desired features in an excellent site. All criteria should be considered with the target audience in mind.

1. **Navigability**
   The learner can move from page to page and item to item with ease and without getting lost or confused. Links are well organised and appropriate link choices are easy to find.
2. **Suitable Hyperlinks**
   The links on a page lead to appropriate pages for the learner to select as their next page. Any choice of hyperlink on a page results in a logical ordering of the material.
3. **Organisation of Hyperlinks**
   Hyperlinks are organised within the page to help the learner make suitable choices.
4. **Instructional Strategy**
   The site has a clear instructional plan. Enough ordering supplied within the hyperlinks and pages to execute that plan. It is as easy, or easier, to use the site as an instructional tool as it is to use it for free exploration. The material presented is not overly repetitive (given the learner's level).
5. **Order of Presentation**
   The order in which concepts are presented is suitable for the particular instructional treatment of the material. (This applies more to the instructional presentation, if one is present)
6. **Organisation of Material (individual pages)**
   The information is clearly labelled and organised in a reasonable fashion. Difficult material is broken into manageable chunks.
7. **Suitability of Content**
   The information is at a level appropriate to the learner's level (understandable, but challenging enough to promote some thought and reflection).
8. **Structure of Topic Emerges**
   A sense of the overall structure of the topic emerges as the learner traverses the site. After using the site, a learner would be able to construct a model of the topic (concept-map, outline, etc). (Assuming they read and understand everything)
9. **Linkages between Concepts**
   Concepts in the topic are connected (via hyperlinks) to reflect the whole topic correctly. For instance, if one was learning about cars, and the sub-concepts were wheels, engine, and pistons, there should be hyperlinks between cars and wheels, cars and engine and engine and pistons. It is less likely that there would be a hyperlink between cars and pistons.
10. **Quantity of Pages**
    The quantity of pages in the site is neither too few nor too many.
11. **Quantity of Hyperlinks**
    There are sufficient links for both backward and forward movement. There are enough links to allow learners to explore the topic and discover the structure of the material.
12. **Quantity of Material (entire site)**
    The topic is adequately covered. The site has a suitable amount of material for the level of the learner.
13. **Quantity of Material (individual pages)**
    Pages have enough material on them, without being overwhelming.

6.5 Website Evaluation Sheet

Site number:
Site purpose and audience: Introduction for novice learner
*Please rate the site on each criterion. Use as scale from 1 to 10 where 1 **indicates a poor** score and 10 **indicates an excellent** score. Refer back to the criteria descriptions as needed.*

	Rating	Comments
Navigability (novice introduction)		
Suitable Hyperlinks (novice introduction)		
Organisation of Hyperlinks (novice introduction)		
Instructional Strategy (novice introduction)		
Order of Presentation (novice introduction)		
Organisation of Material (individual pages) (novice introduction)		
Suitability of Content (novice introduction)		
Structure of Topic Emerges (novice introduction)		
Linkages between Concepts (novice introduction)		

*For the following criteria use a scale from 1-10, where 1 **indicates too few** items, 5 **indicates the correct number** of items and 10 **indicates too many** items.*

	Rating	Comments
Quantity of Pages (novice introduction)		
Quantity of Hyperlinks (novice introduction)		
Quantity of Material (entire site) (novice introduction)		
Quantity of Material (individual pages) (novice introduction)		

Appendix G- Evaluator's Reviews and Comments

Site number: 1

Site purpose and audience: Novice

Please rate the site on each criterion. Use as scale from 1 to 10 where 1 indicates a poor score and 10 indicates an excellent score. Refer back to the criteria descriptions as needed.

	Rating	Comments
Navigability (novice introduction)	3	It was easy to lose track of what I was reading about as I went back and forth
Suitable Hyperlinks (novice introduction)	7	Some of the "top" of page buttons didn't work, which could lead to some problems
Organisation of Hyperlinks (novice introduction)	3	It was difficult to remember how far to go back
Instructional Strategy (novice introduction)	4	Hard to follow
Order of Presentation (novice introduction)	7	Some material was covered in depth several pages after it was first introduced
Organisation of Material (individual pages) (novice introduction)	6	Material was broken into chunks well, but connections between bits should be stronger
Suitability of Content (novice introduction)	3	I would need to already know about the material to follow what is being covered.
Structure of Topic Emerges (novice introduction)	2	It was to easy to lose track of what was going on
Linkages between Concepts (novice introduction)	3	Should be better organised

For the following criteria use a scale from 1-10, where 1 indicates too few items, 5 indicates the correct number of items and 10 indicates too many items.

	Rating	Comments
Quantity of Pages (novice introduction)	8	
Quantity of Hyperlinks (novice introduction)	3	Need more control of how to get to different pages, like where to go next
Quantity of Material (entire site) (novice introduction)	9	A junior high student will be overwhelmed
Quantity of Material (individual pages) (novice introduction)	9	Some pages were far too short, while others were incredibly long!

Site number: 2

Site purpose and audience: Novice

*Please rate the site on each criterion. Use as scale from 1 to 10 where 1 **indicates a poor** score and 10 **indicates an excellent** score. Refer back to the criteria descriptions as needed.*

	Rating	Comments
Navigability (novice introduction)	1	**Where am I going? How do I get back? I don't even know if I saw all the pages!**
Suitable Hyperlinks (novice introduction)	4	**All the "Part 2" sort of stuff makes the links too hard to follow.**
Organisation of Hyperlinks (novice introduction)	2	**Too confusing**
Instructional Strategy (novice introduction)	2	**Kids will get lost**
Order of Presentation (novice introduction)	2	What "order" ?!? It's all over the place.
Organisation of Material (individual pages) (novice introduction)	8	**I do like the material.**
Suitability of Content (novice introduction)	9	**Well written**
Structure of Topic Emerges (novice introduction)	1	**Very difficult to see how it builds**
Linkages between Concepts (novice introduction)	2	**Felt like I was being bounced back and forth between ideas.**

*For the following criteria use a scale from 1-10, **where 1 indicates too few items, 5 indicates the correct number of items and 10 indicates too many items.***

	Rating	Comments
Quantity of Pages (novice introduction)	7	**Too many to keep track of. I didn't know where I'd been, or where I should go.**
Quantity of Hyperlinks (novice introduction)	1	**Can't get back to a specific spot easily**
Quantity of Material (entire site) (novice introduction)	7	
Quantity of Material (individual pages) (novice introduction)	7	

Site number: 3

Site purpose and audience: Novice

*Please rate the site on each criterion. Use as scale from 1 to 10 where **1 indicates a poor** score and **10 indicates an excellent** score. Refer back to the criteria descriptions as needed.*

	Rating	Comments
Navigability (novice introduction)	7	Pretty easy to get around
Suitable Hyperlinks (novice introduction)	7	
Organisation of Hyperlinks (novice introduction)	7	
Instructional Strategy (novice introduction)	8	Easy to tell a kid where to go on site
Order of Presentation (novice introduction)	5	
Organisation of Material (individual pages) (novice introduction)	2	Stuff is all over the place on pages
Suitability of Content (novice introduction)	1	Written well over a junior high level.
Structure of Topic Emerges (novice introduction)	6	
Linkages between Concepts (novice introduction)	6	

*For the following criteria use a scale from 1-10, where **1 indicates too few items, 5 indicates the correct number of items and 10 indicates too many items.***

	Rating	Comments
Quantity of Pages (novice introduction)	6	
Quantity of Hyperlinks (novice introduction)	7	
Quantity of Material (entire site) (novice introduction)	4	Some material is repetitive.
Quantity of Material (individual pages) (novice introduction)	6	

Site number: 4

Site purpose and audience: Novice

*Please rate the site on each criterion. Use as scale from 1 to 10 where 1 **indicates a poor** score and 10 **indicates an excellent** score. Refer back to the criteria descriptions as needed.*

	Rating	Comments
Navigability (novice introduction)	8	**Would be convenient to not have to move linearly all the time... break into categories**
Suitable Hyperlinks (novice introduction)	9	
Organisation of Hyperlinks (novice introduction)	9	
Instructional Strategy (novice introduction)	5	**Small site. You would basically have one shot for a student to go through the whole thing.**
Order of Presentation (novice introduction)	6	Could be linked better
Organisation of Material (individual pages) (novice introduction)	6	**Break into categories that are more managable chuncks**
Suitability of Content (novice introduction)	7	**More images needed on some pages. Good writing for age level**
Structure of Topic Emerges (novice introduction)	6	**Should build up concepts better**
Linkages between Concepts (novice introduction)	6	**Links need to be developed more clearly**

*For the following criteria use a scale from 1-10, where 1 **indicates too few items**, 5 **indicates the correct number** of items and 10 **indicates too many** items.*

	Rating	Comments
Quantity of Pages (novice introduction)	2	
Quantity of Hyperlinks (novice introduction)	5	
Quantity of Material (entire site) (novice introduction)	2	**Cover material in more depth**
Quantity of Material (individual pages) (novice introduction)	5	

Site number: 5

Site purpose and audience: Novice

Please rate the site on each criterion. Use as scale from 1 to 10 where 1 indicates a poor score and 10 indicates an excellent score. Refer back to the criteria descriptions as needed.

	Rating	Comments
Navigability (novice introduction)	2	Need a more simple way to set up links
Suitable Hyperlinks (novice introduction)	3	
Organisation of Hyperlinks (novice introduction)	2	End up bouncing around too much
Instructional Strategy (novice introduction)	2	
Order of Presentation (novice introduction)	5	
Organisation of Material (individual pages) (novice introduction)	5	
Suitability of Content (novice introduction)	6	Some is written above learners level
Structure of Topic Emerges (novice introduction)	6	
Linkages between Concepts (novice introduction)	6	

For the following criteria use a scale from 1-10, where 1 indicates too few items, 5 indicates the correct number of items and 10 indicates too many items.

	Rating	Comments
Quantity of Pages (novice introduction)	5	
Quantity of Hyperlinks (novice introduction)	4	
Quantity of Material (entire site) (novice introduction)	5	
Quantity of Material (individual pages) (novice introduction)	5	

Site number: 6

Site purpose and audience: Advanced

Please rate the site on each criterion. Use as scale from 1 to 10 where 1 indicates a poor score and 10 indicates an excellent score. Refer back to the criteria descriptions as needed.

	Rating	Comments
Navigability (novice introduction)	9	**Broken up in a clear way, but need back buttons on pages**
Suitable Hyperlinks (novice introduction)	9	
Organisation of Hyperlinks (novice introduction)	8	
Instructional Strategy (novice introduction)	4	**Actually says on one page "You already know how to measure the s-p interval"... shouldn't make those sorts of assumptions on a remedial site. Put in links to reinforce these concepts!**
Order of Presentation (novice introduction)	4	Better to put "Plate Tectonics and Earthquakes" first
Organisation of Material (individual pages) (novice introduction)	3	**Some pages are missing material or the material runs off the page.**
Suitability of Content (novice introduction)	7	
Structure of Topic Emerges (novice introduction)	5	
Linkages between Concepts (novice introduction)	3	

For the following criteria use a scale from 1-10, where 1 indicates too few items, 5 indicates the correct number of items and 10 indicates too many items.

	Rating	Comments
Quantity of Pages (novice introduction)	4	**Put in extra pages for remedial material**
Quantity of Hyperlinks (novice introduction)	1	**Need "back" buttons**
Quantity of Material (entire site) (novice introduction)	4	
Quantity of Material (individual pages) (novice introduction)	5	

Site number: 7

Site purpose and audience: Advanced

Please rate the site on each criterion. Use as scale from 1 to 10 where 1 indicates a poor score and 10 indicates an excellent score. Refer back to the criteria descriptions as needed.

	Rating	Comments
Navigability (novice introduction)	4	**Good links on first page, but poor after that**
Suitable Hyperlinks (novice introduction)	4	**Hard to get back to beginning. Why does "mantle" page have "next page" links that take you hopping around topics...**
Organisation of Hyperlinks (novice introduction)	2	**Too much jumping around**
Instructional Strategy (novice introduction)	7	**Easy to get student to where you want from 1st page**
Order of Presentation (novice introduction)	3	Hard to follow jumping around
Organisation of Material (individual pages) (novice introduction)	4	**Runs off page**
Suitability of Content (novice introduction)	4	**Not a lot of coverage for the "Earthquake" material that this page is supposed to be geared towards**
Structure of Topic Emerges (novice introduction)	5	
Linkages between Concepts (novice introduction)	4	

For the following criteria use a scale from 1-10, where 1 indicates too few items, 5 indicates the correct number of items and 10 indicates too many items.

	Rating	Comments
Quantity of Pages (novice introduction)	5	
Quantity of Hyperlinks (novice introduction)	4	
Quantity of Material (entire site) (novice introduction)	5	
Quantity of Material (individual pages) (novice introduction)	5	

Site number: 8

Site purpose and audience: Advanced

Please rate the site on each criterion. Use as scale from 1 to 10 where 1 indicates a poor score and 10 indicates an excellent score. Refer back to the criteria descriptions as needed.

	Rating	Comments
Navigability (novice introduction)	3	
Suitable Hyperlinks (novice introduction)	3	Again, why have all this "Part 3" stuff like on novice site #2
Organisation of Hyperlinks (novice introduction)	2	Too easy to get lost
Instructional Strategy (novice introduction)	2	Hard to get a kid to where I might need them to be on the site
Order of Presentation (novice introduction)	5	
Organisation of Material (individual pages) (novice introduction)	4	Lose sight of how it connects
Suitability of Content (novice introduction)	6	
Structure of Topic Emerges (novice introduction)	4	
Linkages between Concepts (novice introduction)	4	

For the following criteria use a scale from 1-10, where 1 indicates too few items, 5 indicates the correct number of items and 10 indicates too many items.

	Rating	Comments
Quantity of Pages (novice introduction)	3	Need to break up the material onto extra pages
Quantity of Hyperlinks (novice introduction)	9	"Which should I click?"
Quantity of Material (entire site) (novice introduction)	7	Overwhelms a person in its current format
Quantity of Material (individual pages) (novice introduction)	8	Could break up pages more, though, to make them more effective

Site number: 9

Site purpose and audience: Advanced

*Please rate the site on each criterion. Use as scale from 1 to 10 where **1 indicates a poor** score and **10 indicates an excellent** score. Refer back to the criteria descriptions as needed.*

	Rating	Comments
Navigability (novice introduction)	7	
Suitable Hyperlinks (novice introduction)	7	
Organisation of Hyperlinks (novice introduction)	7	
Instructional Strategy (novice introduction)	8	
Order of Presentation (novice introduction)	5	
Organisation of Material (individual pages) (novice introduction)	2	
Suitability of Content (novice introduction)	6	**Better suited to this level than novice**
Structure of Topic Emerges (novice introduction)	6	
Linkages between Concepts (novice introduction)	6	

*For the following criteria use a scale from 1-10, where **1 indicates too few items, 5 indicates the correct number of items and 10 indicates too many items.***

	Rating	Comments
Quantity of Pages (novice introduction)	6	
Quantity of Hyperlinks (novice introduction)	7	
Quantity of Material (entire site) (novice introduction)	4	
Quantity of Material (individual pages) (novice introduction)	6	

Site number: 10

Site purpose and audience: Advanced

Please rate the site on each criterion. Use as scale from 1 to 10 where 1 indicates a poor score and 10 indicates an excellent score. Refer back to the criteria descriptions as needed.

	Rating	Comments
Navigability (novice introduction)	6	Great to have all the links on the first page, but a few pages in a "back" button took me all the way back to the beginning, instead of just back one page. Should be labelled better.
Suitable Hyperlinks (novice introduction)	6	
Organisation of Hyperlinks (novice introduction)	6	
Instructional Strategy (novice introduction)	7	Set out well to allow a person to tell a student exactly where to go.
Order of Presentation (novice introduction)	4	Put "Plate Tectonics Definition" & "Types of Plates" BEFORE covering their "Movement"
Organisation of Material (individual pages) (novice introduction)	6	
Suitability of Content (novice introduction)	7	
Structure of Topic Emerges (novice introduction)	5	Could be more clear
Linkages between Concepts (novice introduction)	4	Need more links between main concepts... although this might serve well as a resource page in its current form.

For the following criteria use a scale from 1-10, where 1 indicates too few items, 5 indicates the correct number of items and 10 indicates too many items.

	Rating	Comments
Quantity of Pages (novice introduction)	5	
Quantity of Hyperlinks (novice introduction)	4	Give me an easier way to go back one page, not just back to the index.
Quantity of Material (entire site) (novice introduction)	5	
Quantity of Material (individual pages) (novice introduction)	5	

Appendix H- Summary of alternate sites generated by APHID

Site Type	NumElements	NumLinks	NumPages	AvgElements	AvgLinks
DFI-Novice	**64**	**58**	**19**	**3.37**	**3.05**
DFI-Int	87	60	19	4.58	3.16
DFI-Advanced	106	60	19	5.58	3.16
Spiral-Novice	**64**	**178**	**50**	**1.28**	**3.56**
Spiral-Int	87	182	51	1.71	3.57
Spiral-Advanced	106	182	51	2.08	3.57
Remedial-Novice	58	77	15	3.87	5.13
Remedial-Int	74	78	23	3.22	3.39
Remedial-Advanced	**92**	**78**	**23**	**4.00**	**3.39**
Review-Novice	41	41	15	2.73	2.73
Review-Int	78	40	15	5.20	2.67
Review-Advanced	58	40	15	3.87	2.67