An Adaptive Domain-Independent Agents-Based Tutor for Web-Based Supplemental Learning Environments

by

Steven Niemczyk

S.B., M.ENG., MASSACHUSETTS INSTITUTE OF TECHNOLOGY (1996)

SUBMITTED TO THE DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY IN THE FIELD OF
INFORMATION TECHNOLOGY
AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
FEBRUARY 2003

Author _____
Department of Civil and Environmental Engineering
October 25, 2002

Certified by _____
Prof. Steven R. Lerman
Department of Civil and Environmental Engineering
Thesis Supervisor

Accepted by _____
Prof. Oral Buyukozturk
Chairman, Departmental Committee on Graduate Studies

To Jessica and my family, for everything

An Adaptive Domain-Independent Agents-Based Tutor for Web-Based Supplemental Learning Environments

by

Steven Niemczyk

Submitted to the Department of Civil and Environmental Engineering
on October 25, 2002, in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the field of
Information Technology

Abstract

The Physics Interactive Video Tutor (PIVoT) is a Web-based multimedia resource for college-level Newtonian mechanics. The Personal Tutor (PT) is an intelligent tutoring system (ITS) integrated into PIVoT, assisting students and teachers in navigating through, understanding, and assessing PIVoT's educational media. PT is adaptive in that it personalizes its functionality to the preferences of its user. The combined PIVoT / PT system was designed to be domain-independent with respect to the style of pedagogy, models of user learning, and instructional algorithms. Thus, this design is easily adapted for use beyond the tested domain of introductory college physics.

PT is designed in the object-oriented paradigm, building upon the recent work in multi-agent systems (MAS). This agents-based approach, along with innovations in negotiating student-agent control and communication, allow current and future competing pedagogical strategies and cognitive theories to coexist harmoniously. New efficient, domain-independent techniques for discovering, updating, and presenting students' contextual interests improve information retrieval and site navigation. Unlike other computer-based instruction systems used as a tool for primary learning and assessment, PIVoT is used as a supplementary resource focusing on providing formative assessment to both student and educator alike. The PIVoT / PT system leverages reusability and system independence, two often-overlooked strengths of agent-based approaches to intelligent tutoring systems. Combined, PIVoT and the Personal Tutor provide an effective proving ground for innovations in intelligent tutoring system design that also reduces the cost of making such software.

Thesis Supervisor: Steven R. Lerman

Title: Class of 1922 Professor of Civil and Environmental Engineering

This page intentionally left blank

Acknowledgements

I would like to acknowledge the Center for Advanced Education Services, the Center for Educational Computing Initiatives, the IBM Fellowship program, and the Singapore-MIT Alliance for generously providing the funding and resources critical to the success of this research. I would like to thank the entire staff of CAES and CECI for their assistance and enthusiasm, which enabled me to continue toward my goal.

I would like to thank Atlantic Philanthropies as the primary foundation supporting the design and creation of PIVoT. I would also like to thank Walter Lewin for his infectious enthusiasm that made the PIVoT and Personal Tutor projects all they could be.

I would like to acknowledge Laura Koller, for her support in all that I did, both at CECI and beyond. Her much-appreciated presence in the audience at each of my shows demonstrates her commitment to me as not just a researcher, but also as a friend.

I would like to acknowledge Phil Bailey for his programming wisdom, guidance, and limitless patience and understanding through difficult and stressful times. I have grown both as a programmer and as a person through the seven years we have worked together, since being an undergraduate researcher at NMIS. I would also like to thank Andrew McKinney for his vast knowledge, his diplomacy, and most importantly, his friendship.

I would like to acknowledge Professor Mitchel Resnick for his time, advice, and guidance. His passion for education was contagious, and markedly shaped my research.

I would like to thank Professor Richard Larson, a member of my committee and the Principal Investigator for PIVoT, for his guidance, support, and direction through this challenging endeavor, as well as the opportunity to work on the PIVoT project. By introducing me to probabilistic reasoning, he not only helped me develop a latent talent, but also changed my approach to my research, solidifying my ideas through the rigors of mathematical thought. Since that first cup of ice cream on the steps of the student center, his confidence and faith in my abilities has been unwavering.

I would like to acknowledge Professor Steven Lerman, for his insightful ideas, invaluable advice, careful guidance, and unyielding support through the seven years I have worked with him. With as little as a sentence, his advice can transform the most frustrating problem into the most facile task. The day he asked to be my Master's thesis advisor changed my academic career. I truly would not be where I am today without him.

I would also like to thank my friends for their support and encouragement. I would like to thank my sister for her loving support. I would like to thank my dad for love and support, and most important to the successful completion of this endeavor, teaching me the value of a day's work. I would like to thank my mom for her love and guidance throughout the years, through good times and bad.

Finally, I would like to thank Jessica for her love and understanding. Her inspiration has driven me to be all that I can, enabling me to complete this long yet rewarding journey.

Contents

Illustrations

Tables

Abbreviations

AI ..... Artificial Intelligence

ALBM ..... Adaptive Learner Behavior Modeling

API ..... Application Programming Interface

CSM ..... Cosine Similarity Metric

CV ..... Context Vector

DB ..... Database

DIO ..... Domain-Independent Ontology

FAQ ..... Frequently Asked Question

HTML ..... Hypertext Markup Language

HTTP ..... Hypertext Transfer Protocol

KB ..... Knowledge Base

IR ..... Information Retrieval

ITS ..... Intelligent Tutoring System

JVM ..... Java Virtual Machine

MAS ..... Multi-Agent System

PIVoT ..... Personal Interactive Virtual Tutor

PT ..... Personal Tutor

TYK ..... Test Your Knowledge

USR ..... UserSession.Request

VSM ..... Vector Space Model

WBLE ..... Web-Based Learning Environment

WS/JVM ..... Web Server / Java Virtual Machine

1 Introduction

The Internet has dramatically changed the way higher education courses are taught. Some courses are offered exclusively through the Internet, fueling an exploding interest in distance education. Other courses use the Internet to augment traditional pedagogy, using the strength of the Web to provide asynchronous delivery of course materials, changing the time and space of learning in today's universities. While some Web-based courseware aim simply to make information available to students, others use innovative and complex artificial intelligence techniques to enhance a student's experience through personalized tutoring.

This thesis proposes pedagogical, architectural, and mathematical approaches to providing a personalized learning environment through domain-independent educational software design. This multidisciplinary approach was implemented and evaluated with the Physics Interactive Video Tutor (PIVoT), a supplementary online multimedia resource used in a freshman mechanics course at the Massachusetts Institute of Technology (MIT), and its embedded intelligent agent, the Personal Tutor (PT). By subscribing to domain-independent principles, this educational software platform can be used in several academic domains, increasing the scope of this research beyond the initial domain of introductory mechanics.

There is no single approach to Web-based pedagogy. As such, the software design proposed and tested here can support multiple pedagogical strategies simultaneously, further increasing the reach of the research described herein.

This chapter establishes the guiding principles and overall results of this thesis. The following sections discuss the motivation for this research, as well as its goals and accomplishments. An overview of the rest of this document is also provided.

1.1 Motivation

The motivation for this research has at its core a need for a combined technological and pedagogical solution to the difficulty in creating low-cost educational courseware. As more courseware projects are created, reusing designs from previous software development efforts minimizes the marginal cost for each project. For this approach to work, the educational software platform must be designed in a way that is generalized enough to work across multiple domains. Since the Internet provides simple means for deploying similar software systems quickly, the impetus for reusable courseware designs comes as no surprise.

1.1.1 Pedagogy and the Internet

There is little doubt that the Internet has altered the nature of instruction in recent years. In higher-education settings and beyond, course materials have often migrated to the World Wide Web (WWW). Colleges and universities around the world are now rushing to bring courses to the Internet, many of which make use of the Web for all or part of course delivery. The quality of current offerings available on the Internet however is quite varied. High-quality educational pages require careful planning, development, and maintenance. Not surprisingly, "the Web is littered with higher education course-related pages that are technically and/or pedagogically flawed." [Maddux and Cummings 2000, 148]

Web-based learning has changed the time and space of education. Students can do course work on their own time, at their own pace, from almost anywhere [Cole 2000, *i*]. Unfortunately, high-quality Web-based educational course materials are expensive to develop, and often with each new course, more costs are incurred. Therefore, it is not hard to see why reusable, pedagogically sound software designs are highly desirable from a pragmatic point of view. This thesis describes one such design approach, and how its usability can be enhanced through an embedded intelligent software agent.

Much of the interest in Web-based instruction is in its application to distance education [Maddux and Cummings 2000, 147]. Distance education has become popular due to the power of Web-based delivery systems, as well as the comfort of learning at home [Vasileva *et al.* 2001, 791]. Initially, most educational Web sites were created

primarily for courses delivered entirely on the Internet; this practice has changed in recent years [Maddux and Cummings 2000].

As educational use of the Web has matured, traditional courses began to make strong use of the Web as well. As the Internet becomes the *de facto* information communication medium, the Web is becoming the center for official student-faculty communication. The Web is used to deliver course materials, leveraging the asynchronous nature of the medium. In addition, some courses use the Web for submission of assignments, taking advantage of the authentication and storage that Web-based submission systems often provide.

A small number of traditional courses are using the Web as a supplementary educational resource. This is in contrast to many Web-based intelligent tutoring systems, which aim to be the primary method of instruction, as in distance education applications. This thesis arose out of one such supplementary education resource, the Physics Interactive Video Tutor.

1.1.2 The PIVoT project

PIVoT is the result of a collaborative venture at MIT between the Center for Advanced Educational Services and the MIT Department of Physics. This venture capitalized on an existing set of video lectures by MIT's renowned Physics Professor, Walter Lewin. This resource, along with the full support of Walter Lewin himself to record additional lectures, demonstrations and problem reviews, allow for an unprecedented opportunity to create an advanced digital-video learning environment for assisting in the teaching of Freshman Newtonian Physics, MIT's Physics subject 8.01.

The Physics Interactive Video Tutor (PIVoT) project allows students to receive help via interactive video and text help sessions. Freshmen have access to an online database replete with instructional video, hypermedia textbooks, and explanatory still graphics and animations. This Web-based learning environment (WBLE) aspires to allow students to feel like they have Professor Lewin as their personal desktop tutor [Larson and Koller 2001]. The project explores whether new digital multimedia technologies can increase by an order of magnitude or more the extent of "face to face

contact" that a student has with her/his professor (although this "contact" still misses some of the elements of in-person interaction).

While the video element of PIVoT helped to augment face-to-face time, it by no means intended to replace the traditional pedagogy of 8.01. Walter Lewin's help videos had been offered for years on MIT's closed-circuit cable networks, 24 hours a day. They would loop repeatedly, every half-hour, on topics relevant at that point in the course. This synchronous delivery mechanism had known limitations: students who fell behind or otherwise had problems with material from earlier in the course would go unaided.

Offering these videos online allows them to be delivered asynchronously, granting students more freedom in when and where they can get video help. Videos can be segmented into shorter topical units that allow students to more directly access the information they need. Students may also access clips from any point in the course, assisting students who need help with older, and often more fundamental, topics.

Besides being a distribution site for help videos, PIVoT offers additional content that is suited for students with learning styles less suited to instructional video. A list of frequently asked questions (FAQs) and their answers provide students with help with common physics problems. An online textbook provides a searchable reference to all mechanics topics. Multiple-choice practice problems are also provided to allow students to assess their own knowledge without affecting their grade. Simulations allow students to experiment with topics best explored in a "hands on" setting, such as elastic and inelastic collisions, projectile motion, and others.

1.1.3 The Personal Tutor

While the vast knowledge base of PIVoT can be searched and browsed in several ways, it is often difficult for students to know precisely what it is they are looking for, as discovered in a three-school study of an early version of PIVoT [Lipson 2001]. The primary motivation for the Personal Tutor was to provide an alternate mechanism for searching through the vast database in PIVoT.

PIVoT usage surveys showed the need for greater guidance through the physics content. In an early version of PIVoT, before the introduction of the Personal Tutor, between 25% to 50% "had difficulty finding the information they were looking for."

[Lipson 2001, 14]. While the existing search mechanism could be refined and enhanced, one might also argue for a collection of alternate means to search through the system. Additionally, while the PIVoT system aspired to be a tutor, it was designed principally as a learning resource. A need for a more tutor-like enhancement to the existing system was needed. Intelligent interface agents, often used to augment difficult to use Web interfaces, were chosen as a viable solution. As discussed in later chapters, the Personal Tutor proved an invaluable part of a more robust PIVoT interface.

The name Personal Tutor was chosen because it provides adaptive help beyond what is designed into its parent, PIVoT. Note that the intelligent agent system is called a tutor, even though its host website, PIVoT, is also called a tutor. From the student's perspective, both PIVoT and the Personal Tutor teach in tandem: The former provides information using a searchable, pedagogical multimedia resource; the latter provides individually tailored guidance via better and easier access to the PIVoT resource.

1.1.4 Motivating Principles

As stated above, there is no single approach to Web-based instruction. The literature also shows that there are many different, successful designs to intelligent tutoring systems, both with and without an intelligent agent approach. In evaluating solutions to student-guidance in Web-based learning environments, several principles, and philosophies guided the research.

In the earliest stages of the PIVoT project, a key goal of the project was to have the design be adapted with minimal effort to other higher-education courses in other domains. A principle of this research is to favor domain-neutral pedagogical approaches (i.e., pedagogies indifferent to academic domain) that adapt well to other courses over more domain-specific approaches, even if the more general approach is less effective.

Personalized Interface Agents have proven useful in reducing disorientation and information overload in user interfaces [Maes 1994]. These assistants have become increasingly part of today's computing environments. A popular and ubiquitous interface agent is the *Office Assistant* found in *Microsoft Office 97* and its successors [Trower 1999]. Sometimes, however, these well-meaning assistants annoy more than they assist; such agents are often ignored or disabled [Huhns and Singh 1998]. A guiding principle

in setting the research goals of this project is to provide an interface agent that is less obtrusive, even if at the expense of reducing its usage.

The literature is replete with different epistemological strategies for Web-based learning. It is also well known that different students respond differently to different pedagogical strategies [Vogel and Klassen 2001, 104]. This research is guided by a motivation to support multiple learning strategies to maximize the type of students that will benefit from PIVoT and the Personal Tutor. This motivates the goals regarding negotiation between agents guided by different strategies that target different learning styles.

1.2 Goals

The primary goal of this thesis is to propose the requirements for – and a possible design of – a domain-independent software platform for pedagogical agent developers. Secondarily, an exemplar system is implemented and evaluated, both technologically and pedagogically. Due to the multidisciplinary nature of such a system, research was conducted into several fields, including: instructional design strategies, pedagogical approaches, computational algorithms, multi-agent infrastructures, and contextual representations of knowledge. This section discusses the goals of this thesis as they related to the various disciplines that shape the design of PIVoT and the Personal Tutor.

1.2.1 Domain Independence

A key theme that runs throughout this thesis is *domain independence*, or *domain neutrality*. While the Personal Tutor and PIVoT were designed and tested on Introductory Newtonian Mechanics, the system was designed to work with little modification in other academic domains, both in physics and beyond. While this enables the PIVoT/PT system to transform many traditional courses with little marginal effort, it also bounds the pedagogical effectiveness because of the restriction to more generalized instructional designs. A key goal of this research is to explore the limitations and advantages of this unique approach to Web-based pedagogy.

Domain-neutrality also applies strongly to the design of the knowledge base (KB) that is central to any intelligent tutoring system. An important goal of this research is to

investigate how newly created, standard metadata representations of educational content can be used in an intelligent way to assist students in self-directed learning.

It is important to note that while the PIVoT system was designed initially to work with MIT's introductory mechanics course, aspirations for expansion to other subjects were present from the beginning of the project. As such, tools and methodologies were developed to simplify future expansion, and another goal of the research was to compare these to what is in use today.

1.2.2 Web-based Pedagogy

The proper design of intelligent tutoring systems requires a foundation in sound instructional-design models. While a full review of computers in education is offered in the following chapter, this section offers a summary of the pedagogical approach in this thesis based on major points from the literature. A key goal of this research is to leverage existing, proven instructional designs, especially those that are successful across multiple domains. For example, while some techniques exist that work exceptionally well in chemistry education, these systems may not translate well to physics, mathematics, or the social sciences. Since it is impossible to find a single pedagogical strategy that will work for all domains, a goal of this research is to identify instructional designs that support the most domains similar to PIVoT's introductory mechanics.

Research has shown that different students use Web-based instructional tools in different ways. A properly designed intelligent tutoring system should adapt to the learning style of the user [Li Xiao *et al.* 2001; McCalla *et al.* 2000]. A major goal of this research is to explore methodologies and algorithms for adapting to individual learning preferences both reliably and intelligently.

In addition to differences in how students use Web-based instructional resources, educators may use the same resource for different instructional methods and cognitive philosophies. This research explores the suitability of a domain-neutral intelligent tutoring system to self-paced and collaborative learning, as well as behaviorist, objectivist, and constructivist epistemological philosophies.

Proper assessment of intelligent tutoring systems provides valuable feedback to educational policy makers, instructors, and students alike. Often collecting assessment

data is expensive and arduous. This thesis investigates automatic techniques that can reduce the difficulty and cost of performing formative and summative assessment. Formative assessment (feedback used for evaluating both student knowledge and pedagogical effectiveness) can be collected continuously throughout the learning process and shorten the improvement cycle in traditional courses. While summative assessment (i.e., feedback used towards grading) can be performed cheaply online, it may discourage students from using these Web resources for their own benefit for fear of affecting their grades.

1.2.3 ITS Architecture

All too often intelligent tutoring systems are created in an *ad hoc* manner. This thesis aims to create a tutoring system designed according to software engineering principles. Toward that end, one goal was to create an Application Programming Interface (API) to simplify the process of creating pedagogical agents.

Since a key goal of this research was to support multiple pedagogical strategies, it became important to choose a design that was neutral to higher-level issues of multi-agent system (MAS) design. In order to support multiple pedagogical strategies, these details are of lesser interest, although current approaches are reviewed in the following chapter. Lower-level design considerations, such as information storage and agent communication, are given greater focus.

An active area of research in multi-agent systems is *negotiation*, or how agents communicate, cooperate, and compete with each other [Zarnekow and Wittig 1998, 41]. This thesis offers a model for agent negotiation that supports the combination of multiple educational experiments. In addition, this research explores innovative yet simple ways to use student feedback in agent negotiation.

In intelligent tutoring systems, another key issue is how control of the learning process is shared between the student and the agent. The Web is essentially a stateless medium, complicating control issues immensely. This thesis aims to overcome these limitations by combining existing software techniques with new approaches.

Often today's pedagogical agents use obtrusive, animated user interfaces that dictate a path through the learning process. In a supplementary Web resource like

PIVoT, it is important to allow the user to explore the Web resource according to self-directed goals [Vogel and Klassen 2001]. This thesis offers a way for students and agents to communicate that balances the need for allowing students to follow their own objectives, with the need for agents to guide students along a different path. At all times, the user has ultimate control and can abandon the agent's plan in favor of his or her own.

1.2.4 Scalability

While every ITS design includes a knowledge base, different domains will vary in the size of these knowledge bases. This thesis aspires to achieve scalability in knowledge base size. The PIVoT system has hundreds of keywords and topics and thousands of educational media items stored and logged within it, and thus the algorithms used must handle knowledge bases of this size and even larger.

Scalability can also affect performance. Over 800 students enroll in a typical fall term of MIT's introductory physics course, 8.01. Considering that at peak times a sizable fraction of these students may be online simultaneously, the system must maintain acceptable performance even when the number of concurrent users rises dramatically.

1.2.5 Platform Neutrality

In order for a Web-based tutor to achieve widespread use, the system must run across a wide-range of clients. Some agents use platform-dependent approaches to deliver a richer experience with animated agents, though at the expense of cross-platform compatibility. A goal of this research is to prioritize a consistent experience and interface across client software and platforms.

Additionally, server technology was chosen to allow migrations to other platforms, as needed. For this reason (and others discussed later), Java was chosen as the platform of choice on the server side. With the cross-platform compatibility of server-side Java, one is able to leverage existing code libraries for intelligent tutoring systems, database management, mathematical algorithms, etc.

1.3 Overview

The following chapter discusses the context in which PIVoT and the Personal Tutor was created through a literature review of the various disciplines relevant to PIVoT. After the background is established, the overall design of the PIVoT and Personal Tutor system is discussed in the third chapter. After the overview, each key component in PIVoT and the Personal Tutor is described in detail in its own chapter. This document then offers the preliminary data from deployments of PIVoT. Finally, a review of the accomplishments of this research follows, including a discussion of future work.

2 Background

Due to the interdisciplinary nature of this research, a careful review of the literature spans several disciplines. Most of the current literature in Web-based pedagogy and intelligent tutoring systems comes in the form of case studies that combine various technologies, software architectures, and instructional design methodologies with sundry cognitive theories, epistemological philosophies, and pedagogical approaches. This chapter offers both a broad overview of the technological and educational theories and approaches used to create Web-based learning environments and intelligent tutoring systems, as well as a detailed review of the current research in WBLE and ITS.

Note that several of the technologies and philosophies germane to this research are interdisciplinary in and of themselves. Several technologies and philosophies have applications well beyond the realm of education technology. For this reason, emphasis will be given to the aspects of these core technologies and theories most related to this thesis. In attempting to place this endeavor in its proper context, trends and overarching themes are observed and in turn used to guide the review where possible.

This literature review begins with a discussion of the philosophies and theories behind Web-based pedagogy, including epistemology, cognitive learning theory, and instructional design. It continues with a discussion of related educational technologies, including artificial intelligence, intelligent agents, and information retrieval. It concludes with a comprehensive survey of current intelligent tutoring systems, which combine these technologies and theories in disparate ways.

2.1 Cognitive Theory in Web-Based Instruction

Epistemology is the branch of philosophy that studies the nature of knowledge, its presuppositions and foundations, and its extent and validity [Colaric 2000]. Where epistemology is the philosophy of *knowledge*, cognitive science is the science of *knowing*. Cognitive science constructs and evaluates models of intelligent systems [Luger 1994, 37], both biological and artificial. Cognitive science concerns itself with understanding and modeling human understanding and knowledge, as well as in modeling knowledge in artificially intelligent systems. As such, cognitive theories have strong implications for instructional design, especially with the advent of the Web as an instructional tool [Miller and Miller 2000].

2.1.1 Learning Theory

There is no shortage of terms to describe cognitive learning theories, including *behaviorism*, *objectivism*, *instructivism*, *cognitivism*, and *constructivism*. Though the exact definition of these theories is rather fluid, this section offers workable definitions for these key terms. It is important to note that none of these terms are monolithic, and often sub-theories exist that leverage aspects of several of these main theories.

2.1.1(a) Behaviorism

Behaviorism is concerned with stimulus-response theories that define learning as establishing an associative link between a particular stimulus and a particular response [Colaric 2000]. The pioneering work of B.F. Skinner brought about an interest in the pedagogical applications of behaviorism [Todd and Morris 1995]. Behaviorists do not study introspection (what people think); they only study behaviors because they believe one cannot know what other people think so one should not try to study it and make inferences. Behaviorists believe the only way to get reliable data is to study only what is observable, treating mental processes as a "black box." In order to develop a reliable and useful theory of learning then only reliable and useful data (i.e. observable data) should be used.

2.1.1(b) Cognitivism

A second learning theory is cognitivism. Cognitive psychology arose from a reaction to behaviorism. Cognitivists felt that behaviorism's emphasis on the link between a stimulus and a response was not sufficient to account for all human activity.

Cognitive psychology focuses on *mental processes* that operate on stimuli presented to the perceptual and cognitive systems and which usually contribute significantly to whether or not a response is made, when it is made and what it is. Behaviorists claim that such processes cannot be studied because they are not directly observable and measurable, cognitive psychologists claim that they *must* be studied because they alone can explain how people think and act the way they do.

Research in cognitive psychology still draws from a behaviorist view – *objectively observable behavior is all even cognitive researchers have to go on* – mental representation and processes have to be studied indirectly and researchers draw conclusions about them by inference rather than from direct measure; strict cognitive psychology still tends to adhere to experimental methodology.

2.1.1(c) Objectivism

Objectivism, which some people sometimes refer to as instructivism, separates knowledge from the knower. Because knowledge (i.e., reality or truth) is an external entity, it has structure that can be known objectively. According to Miller and Miller [2000], objectivists treat personal experience or bias as hindrances to accurate perception of reality. As such, appropriate means such as the scientific method can be used to enable accurate perceptions of reality. In education, this translates instructional design into a task of creating a proper symbolical representation of this externalized knowledge so that learners can accurately acquire its meaning.

2.1.1(d) Constructivism

A theory rapidly gaining prominence in instructional design is constructivism. Constructivism emerged out of the research of Jean Piaget, a psychologist studying human development in the 1920s [Wadsworth 1996]. Constructivists believe that learning is an active process of constructing, rather than acquiring, knowledge and that

the goal of instruction is to support that construction rather than trying to transmit knowledge [Colaric 2000]. Constructivism focuses on the construction of new knowledge that is unique to each person as well as the importance of the environment in determining the meaning of reality. Because constructivism has origins as an epistemological approach, "its translation as a valid theory of instruction has not been established." [Miller and Miller 2000, 162]

2.1.2 Learning Theory in Instructional Design

Although competing philosophies about reality, knowledge and truth seem "ephemeral and remote" [Miller and Miller 2000] to designing effective Web courseware, epistemology is at the core of the instructional process. By influencing learning and instructional theory, epistemological beliefs suggest and shape Web technologies. As stated by Miller and Miller [2000], the latter half of the 20th century witnessed a shift toward constructivism, and this shift has strongly influenced the instructional practices used in Web technology.

According to Miller and Miller [2000, 161], several factors influence Web-based instruction. Most importantly, theoretical orientation [2000, 162] affects the intended purpose of presented information. Objectivist designed Web sites serve merely to deliver information, regardless of a student's background knowledge. Constructivists aim to allow students to construct their own interpretation and relation of content, giving students a more active role in the use of the Web. In addition, the intended learning goals of the designers are an important factor in Web-based instructional design. Some sites intend only to impart knowledge, while other sites attempt to apply "knowledge in thoughtful action" [2000, 165]. Finally, learner characteristics affect the instructional design of Web sites. Research has shown that educators must take into account the cognitive characteristics, level of self-motivation and social context of the users of a Web site in order to create effective instructional designs [Abbey 2000; Miller and Miller 2000; Vogel and Klassen 2001].

2.2 Artificial Intelligence in Education

Part of the challenge of artificial intelligence includes the challenge of defining it [Nwana and Ndumu 1998, 29]. It is difficult to define artificial intelligence (AI) without semi-circular reasoning about what is intelligence itself. Nilsson [1998] acknowledges this in defining AI broadly as the discipline "concerned with intelligent behavior in artifacts." Winston more directly defines artificial intelligence as "the study of the computations that make it possible to perceive, reason, and act." [Winston 1992, 5] He continues by distinguishing artificial intelligence from the related disciplines of psychology and computer science, due to AI's emphasis on computation over the former, and perception and reasoning over the latter.

The goals of artificial intelligence are both engineering and scientific in nature [Winston 1992, 6]. The scientific goal of AI is to determine which ideas about knowledge, its representation, and its use, explain intelligence. The engineering goal of AI is to solve real-world problems using the scientific discoveries about knowledge and its representation described above.

Although artificial intelligence has already produced practical and useful systems, the ultimate goal of achieving human-level intelligence is still quite far away [Nilsson 1998]. That being so, there is strong debate among researchers about the best approach to AI. Over the last half-century, many different paradigms have emerged. These paradigms may be clustered into two major groups: classical AI, and nouvelle AI. The rest of this section defines these approaches, discusses popular models relevant to education in each approach, and reviews trends in AI today.

2.2.1 Classical AI

Classical AI achieves intelligence through explicit symbolic representations of knowledge and the world [Nilsson 1998]. This approach to AI represents "knowledge" about a domain in declarative sentences based on or equivalent to first-order logic. Logical reasoning is used to deduce consequences and results from this knowledge. This paradigm has many variants, including several that emphasize the role of formal axiomatization of domains in logical languages. As such, classical AI requires

substantial knowledge of the domain at design-time, and thusly this paradigm is referred to as a *knowledge-based* approach.

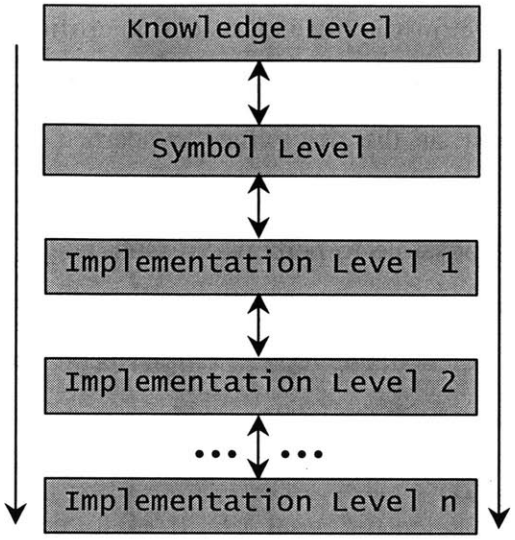

Figure 1: Classical (or "Top-Down") AI

In the symbol-processing approach of classical artificial intelligence, behavior is achieved through pre-defined interactions of several levels in the system. The top is known as the *knowledge level* where the knowledge needed by the system is specified. Below this is the *symbol level*, where the knowledge is represented in symbolic structures. Often these structures are symbolic lists, easily represented in the popular AI language LISP. Most symbol-processing approaches use a "top-down" design strategy, beginning at the knowledge level, proceeding to the symbol level, and continuing downward to implementation levels (see Figure 1 above). This being so, classical AI is often referred to as "top-down" AI.

2.2.2 Nouvelle AI

Nouvelle AI achieves intelligence in a radically different way. This new approach arose out of a frustration with the limitations of classical AI. While classical AI could be used to produce intelligent systems that could perform sophisticated tasks such as playing chess, it had significant trouble with performing basic human functions such as walking. Proponents of nouvelle AI have observed that human intelligence evolved after billions of years of evolution, and as such they claim that in order to make intelligent

machines, one ought to concentrate first on duplicating the signal-processing abilities and control systems of simpler animals, such as insects.

Instead of the complex monolithic closed systems that rarely interact with their environment found in classical AI [Zarnekow and Wittig 1998, 36], nouvelle AI systems consist of simple modules that interact under simple rules with each other and their environment. The intelligence of nouvelle AI is not defined completely before at design-time; rather it is said to *emerge* from these interactions within the system and with the ever-changing environment. As put by Pattie Maes [1990, 1], a proponent of nouvelle AI and one of the first researchers into intelligent agents:

> The functionality of an agent is viewed as an emerging property of the intensive interaction of the system with its dynamic environment. The specification of the behavior of the agent alone does not explain the functionality that is displayed when the agent is operating. Instead, the functionality to a large degree is founded on the properties of the environment. The environment is not only taken into account dynamically, but its characteristics are exploited to serve the functioning of the system.

Nouvelle AI approaches are often referred to as "bottom-up" AI, since these designs tend to start at the lowest level and work upward. At these lowest levels, that concept of *symbol* is not as relevant or appropriate as the concept of *signal.* For this reason, nouvelle AI is said to follow a "subsymbolic" approach.

2.2.3 Models in Education

There are many models of artificial intelligence, both classical, nouvelle, and hybrid. This section offers a brief overview of models used in Artificial Intelligence that have strong applicability to education and ITS design. Expert systems use classical AI to draw inferences from knowledge in a specific domain. Probabilistic Inference and Bayesian Networks are used to model uncertainty, and Markov Modeling and Decision Theory is used to make decisions and observations from this uncertainty.

2.2.3(a) Expert Systems

Rule-based expert systems (also known as *rule-based systems* or simply *expert systems*) apply classical AI reasoning techniques to facts and rules about a specialized field. Expert systems contain four key components: a *knowledge base* consisting of

logical rules about a domain, a knowledge-acquisition system which propagates the knowledge base with these rules, an *inference engine* which uses logic and the predicate calculus to apply these rules to given conditions, and an *explanation system* to communicate these ideas to a user.

Knowledge engineers work with experts in the domain by asking them about situations in search of similarities between them. As such, two key heuristics enable knowledge engineers to acquire knowledge: firstly, be specific as possible, and secondly look for outwardly similar appearing situations and find distinguishing characteristics between them [Winston 1992, 168].

A survey of applications of expert systems highlights domains such as medicine, engineering, and business [Nilsson 1998, 280]. The canonical example of (and motivating force behind) an expert system is an automatic medical diagnosis system [1998, 280], and applications with similar usage metaphors are common. Expert systems are used in education because they can be used to diagnose student knowledge and help detect misconceptions in student logic [Prentzas *et al.* 2001].

2.2.3(b) Probabilistic Inference and Bayesian Networks

Often artificially intelligent systems only have uncertain information about its task and its environment. As such, probability theory is often used in the design of artificial intelligence systems. Often, the certainty of one unknown affects the certainty of other unknowns. Additionally, new information about the task or environment, which may itself be uncertain, alters existing (*a priori*) probabilities to create a new (*a posteriori*) view of the world. This kind of reasoning with uncertainty is known as *probabilistic inference*, or *Bayesian* inference due to Bayes' theorem, which relates *a priori* to *a posteriori* probabilities.

Bayesian networks are convenient structures for representing probabilistic inference. Bayesian networks are also referred to as *belief networks*, because they can be used to represent an intelligent system's uncertain "beliefs" about the environment in a formally correct way. A Bayesian network is a directed, acyclic graph (DAG) where each node represents an unknown. Edges in the graph represent causal relationships between unknowns, and thus such networks are sometimes caused *causal networks*.

Bayesian networks offer a mathematically formal alternative to less formal "fuzzy logic" models found in artificial intelligence. Bayesian networks have strong pedagogical applications because of their use in modeling student knowledge. Due to the inherent uncertainty of assessment and other data gathered by a tutoring system, belief networks are quite popular in ITS designs.

2.2.3(c) Markov Models and Decision Theory

Decision making with uncertainty is often necessary in AI applications. As discussed above, belief networks reflect the uncertainty of the environment, and thus statistical models are needed to simplify the decision making process. *Decision theory* is the discipline of evaluating behavior models stochastically to arrive at optimal decisions. Because real world processes are often complex and time-dependent, stochastic models often use simplifications that approximate reality. One simplifying approximation is the *Markov assumption.* The Markov assumption states that actions or decisions based on the current state are made independently of future and past states. While this assumption is rarely true in the real world, when used properly, this assumption can vastly simplify computation and interpretation of carefully defined models. Models that adhere to this design assumption are called *Markov models.*

Markov models are state transition diagrams with probabilistic transitions that use the Markov assumption, that is, each state's transition probabilities are independent of how one arrived in that state. These models can have applications in understanding student behavior by revealing how students interact with a computer interface and draw conclusions about how to make it more effective [Branch *et al.* 1999]. Hidden Markov models (HMM) allow for greater modeling freedom by allowing the observation of state to be "hidden," and thus uncertain and probabilistic itself [Jung-Jin Lee and McCartney 1998]. While HMMs had origins in speech recognition, these models can be used in behavior modeling as well, allowing for optimal decisions about behavior under even greater uncertainty. A more detailed review of Markov and Hidden Markov models that emphasize its applications to this research is offered below in section 6.3.

2.2.4 Trends

The field of artificial intelligence has changed substantially over the years. Winston [1992] observes that older ideas have been displaced with newer ones, and technology often plays a part. As computing environments experience exponential growth in speed and memory correctly predicted by Moore's law, approaches that seemed unthinkable over a decade ago are valid today. For example, it is practical today to control a robot arm using a fast lookup table, where complex logic would have been required in the past [Winston 1992, 397].

While the debate about the applicability of classical and nouvelle AI to solving large problems rages on, several researchers have found success in producing hybrid approaches that combine classical and nouvelle approaches to AI [Nilsson 1998, 7]. Distributed Artificial Intelligence (DAI) uses symbolic reasoning combined with the nouvelle AI concept of individual "situated" components cooperating and competing with each other [Haugeneder and Steiner 1998]. As such, solutions to problems with DAI *emerge*, despite the use of classical symbolic approaches within each agent. The canonical example of a DAI system is the multi-agent system (MAS), discussed in detail in section 2.3 on Intelligent Agents, below.

2.3 Intelligent Agents

Intelligent agents and agent-based systems form new paradigms for developing software applications [Jennings and Wooldridge 1998]. While agent-based systems are of intense interest to researchers in computer science and artificial intelligence, the terminology used to describe these systems, including the definition of the word *agent* itself, is not universally understood [Jennings and Wooldridge 1998; Nwana and Ndumu 1998; Brenner *et al.* 1998]. Nevertheless, a workable definition for agents and agent-based systems is presented here, along with a vocabulary for understanding agent-related concepts and characteristics.

An intelligent agent can be thought of as any entity capable of performing tasks on the behalf of a user or contracting party for which intelligence and specialized knowledge of a domain is required [Zarnekow 1998a]. Although this definition could be used to describe human and hardware agents (i.e., travel agents and robots), emphasis

here is given to intelligent software agents. As such, unless otherwise stated, the terms *agent*, *intelligent agent*, and *intelligent software agent* will be used interchangeably. An alternative definition for agent is any system capable of interacting independently and effectively with its environment via its own *sensors* and *effectors* [Chien-Sing Lee and Singh 2001, 235].

An *agent-based system* is any system in which the key abstraction in its design is an agent, as defined above [Jennings and Wooldridge 1998, 5]. It is important to note that agent-based systems may contain any non-zero number of agents. The multi-agent case is intuitively more complex, due to the issues regarding *negotiation* between agents, but preferable for certain problems. These systems are referred to in the literature as multi-agent systems (MAS). The single-agent case is also common, and often adequate for many tasks, such as the class of systems known as *expert assistants* [1998].

2.3.1 Characteristics and Classifications

Agents can best be understood through the fundamental characteristics that describe them. Firstly, agents are *situated* in an environment (i.e., the Internet) with which they can interact *responsively*, *proactively*, and *socially*. Agents are *responsive* in that they can perceive their environment, detect changes within it, and take appropriate actions in a timely fashion. Agents are *proactive* in that they do not simply react to changes in their environment but rather take the initiative when appropriate, demonstrating goal-directed *behavior*. Agents are *social* in that they interact with their users, other agents, or both to solve problems and achieve their goals.

Agents and agent-based systems are often classified and categorized in several ways, yet these distinctions are not definitive as well [Nwana and Ndumu 1998, 30]. Agents are best categorized and analyzed multi-dimensionally [1998] and this section offers both the fundamental characteristics of agents, as well as common characteristics used to provide a broad understanding of agent research.

2.3.1(a) Deliberative and Reactive Agents

Agents are described as being either *deliberative* or *reactive*, depending on the presence or absence of an internal symbolic reasoning model, respectively. Deliberative agents assume an explicit symbolic representation of their environment, maintaining the

traditions of classical AI. Reactive agents have their roots in the more decentralized nouvelle AI, and provide increased dynamic flexibility at the expense of complex reasoning processes.

Deliberative agents normally create a model of their environment in advance, and this model becomes the main component of the agent's knowledge base. Creating an internal model elaborate enough to achieve an adequate amount of functionality requires a careful design and ample resources. Additionally, one must be able to anticipate at design time errors and irregularities that might occur. Not surprisingly, deliberative agents have only limited suitability for use in dynamic environments [Zarnekow and Wittig 1998].

In addition to their internal symbolic model of their environment, deliberative agents have the ability to make logical decisions from knowledge stored in their internal state. Such agents have symbolic representations for beliefs, desires, goals, intentions and plans, and an architecture is designed to use this "mental model" to make appropriate decisions about which actions (if any) to take. Often these agents follow so-called BDI architectures, short for *beliefs*, *desires*, and *intentions* [Lind 2001, 13].

Deliberative agents have a range of problems, often reflecting the problems of classical AI itself. The chief criticism of deliberative agents comes from their rigid structure. Software agents, especially those situated on the Internet, have too dynamic an environment to make effective use of a symbolic model designed before run time. Often a workable model of a complex situation (such as the relationship between an agent and its "real" environment) cannot be fully understood at design time. In addition, adding this data at run time is difficult as well. As stated by Zarnekow and Wittig [1998], "because the necessary knowledge and the required resources are not normally available, it is difficult for such agents during their execution to add in their existing model new information or knowledge about their environment."

Reactive agents stand in sharp contrast to, and came as a reaction to the philosophy of, deliberative agents. Lacking an internal symbolic model of their environment, these agents obtain their intelligence and behavior not from a centralized, complex reasoning process based on internal knowledge, but from the complex interactions of simple stimulus-response rules [Nwana and Ndumu 1998, 30].

Reactive agents need not have a complex structure to navigate within a complex environment [Zarnekow and Wittig 1998, 50]. Instead, reactive agent designers observe simple principles or dependencies in the environment. Such an approach has a more granular approach to its modularity, producing task-specific *competence modules* that are easy to design and correctly implement. This decentralized approach serves to increase the fault tolerance and robustness of reactive agents, making them highly suitable for dynamic environments such as the Internet.

Reactive agents are not without their own set of problems. It is much harder to make reactive agents demonstrate goal-oriented behavior. Purely reactive agents also do not possess any capabilities to create plans. Additionally "the extent to which [the inability to create plans] negatively affects its tasks to be performed cannot be fully determined." [1998, 52]

It is important to realize that modern systems, including the ones mentioned later in this review, are neither purely deliberative nor purely reactive. It is common to design hybrid systems that leverage the advantages of both approaches and integrate them into a common platform. The Personal Tutor is an example of such a hybrid system.

2.3.1(b) Static and Mobile Agents

Agents may be classified as either static (stationary) or mobile. Mobile agents have the ability to move freely around the network in which it resides, where static agents are bound to a single computer. Static agents may send messages to other agents via a network, but each agent resides and runs exclusively on a single machine. Mobile agents can clearly pose a security risk in open networks such as the Internet, and are uncommon in the literature on pedagogical agents. Thus, stationary agents are emphasized here.

2.3.2 Collaborative Agents

According to Nwana and Ndumu [1998, 32], *collaborative* agents emphasize autonomy, as well as cooperation and negotiation with other agents, in order to perform tasks for their owners. Collaborative agents aim to solve problems too large for a centralized agent system, or to allow for the interconnection of existing legacy systems. These agents tend toward large, static coarse-grained agents [1998, 32]. While these

agents have applications in organizational decision-making or industrial settings, they are not seen as applicable to pedagogy, and as such are not emphasized in this review.

2.3.3 Interface Agents

Interface agents emphasize autonomy and learning in order to perform tasks for their owners. [Nwana and Ndumu 1998, 35] These are agents used in simplifying, enhancing, and assisting user interactions with existing applications, and thus interface agents adhere to the metaphor of *personal assistant* [Maes 1994]. Interactive agents provide *proactive* assistance by observing and monitoring a user's actions and suggesting better ways of doing the task.

Maes [1995] observes the two key preconditions necessary for creating adequate interface agents: firstly, that there exist repetitive behavior (necessary for the agent to learn from) and, secondly, that this behavior varies for different users. Many pedagogical applications where agents are useful meet these preconditions, and thusly, this research proposes an interface agent approach to pedagogical Web sites.

According to Nwana and Ndumu [1998, 35-36], there are three general benefits to interface agents. Firstly, they reduce the workload for both the end user and software developer. Secondly, the agent's ability to adapt to a user's preferences and habits increases the comfort of the user. Thirdly, the knowledge gained about the habits and preferences of users can be shared with developers and the users, as appropriate. Interface agents have been used to assist in meeting scheduling [Kozierok and Maes 1993], Web browsing [Lieberman 1995], and news filtering [Maes 1994]. Interface agents are excellent for real applications because they are simple, operate in a limited domain, and do not require cooperation.

2.3.4 Pedagogical Agents

Pedagogical agents are software agents used for teaching or tutoring purposes. Animated pedagogical agents are a common sub-class of *embodied* pedagogical agents. *Embodied* pedagogical agents have anthropomorphic traits, including emotion, natural language ability, and personality. Animated pedagogical agents display their embodiment through a cartoon-like interface [Maes 1995; Trower 1999; Mitrovic and

Suraweera 2000; Elliott *et al.* 1999]. Since they are often used in tutoring the user, this application of agent technology is discussed at length in section 2.5 on intelligent tutoring systems.

2.4 Information Retrieval

Information Retrieval, known as IR in the literature, is a branch of Information Science dealing with document and media retrieval from information systems such as databases and catalogs. Not surprisingly, the advent and rapid expansion of the World-Wide Web has renewed interest in and transformed this discipline. Information Retrieval is used in intelligent tutoring systems to provide formally correct ways of finding the most relevant information to student queries.

2.4.1 Vector Space Model

In Information Retrieval, the predominant representation of relevance between documents and queries is a geometric one. The vector space model (VSM), pioneered by Salton [1975] treats all keywords in a collection of documents as dimensions in a *document space.*

In VSM, each document is represented by a vector, with weights at each dimension corresponding to the presence or absence of each keyword. Present keywords will have non-zero coefficients proportional to the prominence of each keyword in the document. Absent keywords have a coefficient of zero. In practice, these coefficients are either binary indicators (1 for presence, 0 for absence) or measurements of keyword frequency. Queries made to document retrieval systems are also represented as a vector. Each keyword present in both the query and the document space corresponds to a non-zero entry in the query's vector.

2.4.2 Cosine Similarity Metric

The cosine similarity metric (CSM), also introduced by Salton [1975], compares documents and queries for similarity by measuring the cosine of the angle between them. As can be shown geometrically, this is simply the normalized dot product of the two vectors, and is a value between zero and one, corresponding to the similarity between the

document and the query. All documents in the document base are evaluated for similarity with the query and then sorted by this metric. The documents with similarity measures closest to one are retrieved and presented to the requestor in decreasing order of similarity. An example using CSM is shown in section 4.2.2(c) below.

2.4.3 Applications

Most modern Web-based search engines use VSM and CSM in their retrieval systems. While more sophisticated techniques are used for clustering documents by relevant keywords or reducing the dimensionality of the document space, the principal components of CSM and VSM remain intact.

An accepted and common application for agents is information management [Jennings and Wooldridge 1998, 12]. By assisting in retrieval and filtering, agents help reduce the information overload often experienced today [Maes 1994]. Since information overload is often most pronounced on the Web, research into personalized Web agents that reduce information overload is common. Junggee Han and Juntae Kim [1999] created such an agent using the vector space model along with Bayesian classification techniques to build a *content profile* for recommending both content and URLs. Implicit feedback is used by monitoring the bookmarks made by the browser. The cosine similarity method was used for ranking items by relevancy, and since no explicit feedback was necessary, this agent minimizes user intervention. Young *et al.* proposes a similar agent [1999] using keyword extraction from Web-documents with VSM and CSM.

Dimitrov and Warren [1999] use agents to minimize the impact of "poorly constructed queries" by using the basic VSM model to represent through keywords a user's long-term goals. Standard searches are enhanced by remembering keywords used from the Web-based searches of past sessions. A similar approach is taken by Jee-Haeng Lee and Cho [1999], evolving vector representations of user search goals using agents programmed via the AI technique of genetic algorithms (GA).

2.5 Intelligent Tutoring Systems

The field of intelligent tutoring systems (ITS) is intrinsically multi-disciplinary, requiring research efforts in several disparate domains [Kam-Wah Wu 1993, 97]. Computer science provides insight into the computational aspect of ITS. Cognitive science reveals information on cognition and epistemological issues. Behavioral psychology helps on the understanding of learning behavior. Educational studies investigate the effectiveness of different pedagogical approaches. This section offers a brief history of early tutoring systems, followed by a categorized review of the current case studies in ITS design and practice.

Intelligent Tutoring systems have several common components, usually found in all such systems [CoMPIO 2001a]. Firstly, the *student model*, used to measure the progress of the student throughout a session or across multiple sessions. Secondly, the *knowledge base*, where information about the domain is stored. Thirdly, the evaluation module, which given some understanding of the student's performance and interests, decides what to do next. Fourthly, the *pedagogical rules*, recognized as the area of "greatest divergence" [2001a] across intelligent tutoring systems. The final common component of tutoring systems is the *learning theory* of the system, which may include constructivism, instructivism, cognitivism, or others.

Most of the conceptual foundations used in today's tutoring systems were developed before 1990 [2001a], with current systems expanding and combining seminal principles from these early intelligent tutoring systems. The following sections each analyze a particular aspect of intelligent tutoring systems research. Each section offers both the seminal work in that area as well as a survey of current research on that aspect.

2.5.1 Instructional Strategy and Domain Dependence

The first ITS was SCHOLAR, designed in the early 1970's to teach South American geography [CoMPIO 2001a]. Its principal pedagogical strategy was to use the Socratic dialog. It supported the *mixed initiative* approach, where both students and the system may ask questions during the learning sessions. Early ITS systems emphasized duplicating the structure of student/human-tutor interactions, with limited success, primarily due to the technological challenges of making a computer system intelligent

enough to field all kinds of questions (interrogative, interpretive, assertive) in a pedagogically useful and conversationally normal manner [Graesser *et al.* 1993].

The advent of the Web changed pedagogical approaches to educational software, focusing on simpler instructional approaches that are more easily computerized, and consequently, less similar to human tutoring models. Web-administered multiple-choice tutors have shown promise because of their ease of development [Hoole *et al.* 2002], use in grading, and effectiveness in offering students immediate feedback on the understanding of their progress [Buchanan 2000]. Web-based quizzes also provide valuable assessment data while leveraging the asynchronous nature of the Internet [O'Sullivan 2000, 62].

Many intelligent tutoring systems create complex instructional strategies requiring extensive expert system representations customized to the knowledge domain, and consequently, substantial input from knowledge engineers. Such designs have shown success in particularly conceptually challenging domains such as geometry [Aleven and Koedinger 2000], physics [Albacete and VanLehn 2000], and computer programming [Mitrovic and Suraweera 2000]. It is important to note that these systems are quite domain-dependent, and as such, lessons and successes from one system cannot easily be applied to others.

Very few attempts have been made to create software agent systems that are domain-independent in instructional design. While domain-independent instructional algorithms [Prentzas *et al.* 2001] or communication frameworks [Paiva *et al.* 1999] exist, actual domain-independent systems are almost completely absent from the literature.

Masthoff and Van Hoe [1996] describe one attempt at such a domain-independent ITS, based on simple general rules, but such a system was purely reactive, lacking any planning mechanism or central control to guide a student through the curriculum or suggest appropriate content based on context [Les *et al.* 1999]. This system also failed to leverage the standards made available through Web-development.

2.5.2 Learning Content and Metadata

In recent years, there have been attempts to create domain-independent standards for representing courseware, also referred to in the literature as *learning content*. To

achieve this goal, information about content, known as *metadata*, must be standardized to facilitate interoperability with other users. While standards now exist, the confusion of many differing standards has slowed adoption, and thus hindered the evolution of domain-independent ITS design [Mühlhäuser 2000].

The path to a unified learning content standard is sinewy. Early standards such as the Dublin Core [Weibel *et al.* 1998] were created for all Internet content, without targeting the specific needs of educational content. The Instructional Management Systems (IMS) project in 1998 built upon the Dublin Core to create the IMS metadata standard, specifically designed for learning content [IMS 2001]. With other competing groups such as the Aviation Industry CBT Committee creating its own CMI (Computer Managed Instruction) standard [AICC 1998], the need for a unifying standard arose. In 1999, the United States Department of Defense created the Advanced Distributed Learning (ADL) initiative. The ADL created the *Shared Content Object Reference Model*, known as SCORM [ADL 2001], providing a single standard for educational content developers. Whether such a standard achieves its lofty goal, however, remains to be seen [Nakabayashi *et al.* 2001].

In recent years, Carnegie Mellon University's Consequence Management Program Integration Office has used this model to create domain-independent systems that use metadata standards for creating intelligent tutoring systems. Such a system may hold the promise of "'evolving' ITS systems that are not constrained to any single instructional method" [CoMPIO 2001b] or domain [CoMPIO 2001c], though this approach to ITS design is rather nascent.

2.5.3 Cognitive and Pedagogical Approaches

There are many different cognitive and pedagogical approaches to ITS design. They address issues of student versus tutor control, how to apply cognitive theories of learning, and the role of assessment in intelligent tutoring systems. As stated above, there is great divergence in cognitive and pedagogical approaches. This section offers a short review of recent work.

Several studies conducted in recent years have revealed that introductory physics students have more difficulty solving qualitative problems than quantitative ones. In

order to improve the instruction of the subject, Albacete and VanLehn [2000] created the Conceptual Helper, part of the Andes project at the University of Pittsburgh [Gertner and VanLehn 2000]. The Conceptual Helper follows the model-tracing paradigm. That is, the tutor attempts to model the thought process of the student as he or she steps through the problem. While the system has been shown to be successful, it is strongly tied to the unique problems of physics education. One of the main goals of the Conceptual Helper is to solve common misconceptions about physics, and while there is a lot of literature to support the validity of this approach, it is not well adapted to use beyond physics.

The level of intelligence in intelligent tutoring systems is also a matter of great debate. Aleven and Koedinger [2000] studied the use of intelligent and "dumb" help systems in a 9^{th} grade geometry ITS and found that students often lack the meta-cognitive awareness to seek help when needed. While they suggest forcing students to use help even when they do not solicit it, other researchers emphasize the importance of student control in ITS use, to avoid information overload or disorientation [Vogel and Klassen 2001; Thomas and Rohwer 1993; Chien-Sing Lee and Singh 2001].

On the level of intelligence in tutors themselves, Baylor [2001] observes that "More intelligence is not necessarily better from a pedagogical perspective." Boulay shows [2000] that while unintelligent versions of educational software produce marked improvements over traditional pedagogies, the intelligent versions of the same software improve results more dramatically. Furthermore, increasing the knowledgability of the tutor decreased the number of steps necessary to solve problems. Boulay notes, however that far more time is spent on intelligent v. non-intelligent versions, which may not only bias the result but be a strong negative in time-pressed educational settings.

2.5.4 Assessment

While ITSs can be used to assess the performance of students, there is debate over what kinds of assessment data tutoring systems should emphasize in collection. The emphasis of one approach over another reflects the cognitive and pedagogical assumptions of their designers.

Assessment is defined as a process for obtaining information that is used for making decisions about students, curricula and programs and educational policy [Colaric

2000]. Assessment is critical not only in evaluating the students' knowledge before, during, and after the learning activities, but also in evaluating the educational activities themselves. There are two major kinds of assessment found in ITS design: *summative assessment* and *formative assessment.* Summative assessments are used to formally assess whether students have achieved learning targets, including assigning grades. Formative assessments are used to help improve students' achievement of learning targets. Used by instructors and designers to plan learning activities and to diagnose student misconceptions, it needs to be gathered while the instruction is ongoing.

Providing feedback to students rather than evaluating them for course grades, the value of formative assessment is well known [William and Black 1996]. The function of formative assessment is essentially to assist learners in "closing the game between actual and desired levels of performance." [1996, 543] Many challenges exist in computerization of formative assessment in intelligent tutoring systems. Feedback must be given at an appropriate point in the learning process. For example, there is little point in giving a student feedback when there is little or no time to act on it [Buchanan 2000, 194]. The literature has also shown that timely feedback is key to its value [Baylor 2001]. To be useful in providing immediate feedback, computerized formative assessment must be gathered in an efficient and quantifiable manner [Buchanan 2000].

One form of assessment that is conspicuously absent from ITS designs in the literature is automatic *program assessment.* Program assessment is used to evaluate the effectiveness of the system or instruction itself. Traditionally such assessment is done by hand, through surveys, meetings, etc., offline from the tutoring process. An interesting avenue of research is to create automatic program assessment systems into the ITS design that leverages what computational systems do best: collect and numerically analyze data.

2.5.5 Learner Modeling

In the early 1980s, a tutor for the programming language LISP was created at Carnegie Mellon University by J. R. Anderson, based and the ACT* model of cognition [Anderson 1983]. ACT* identified *procedural* and *declarative* knowledge as part of its expert problem-solving model. ACT* was seminal for introducing the *model tracing* paradigm, whereby the system responds with hints to divergences students make from the

path of the expert model. Several ACT* based tutors were created for other domains since the early 1980s, including Pascal, algebra, and geometry [CoMPIO 2001a]. Today, several other systems use model tracing to control student-tutor interactions [Albacete and VanLehn 2000; Peng-Kiat Pek and Kim-Leng Poh 2000].

Model tracing, while an effective approach to creating a student model of his or her knowledge, is an extremely difficult task due to the search spaces involved [Belkada *et al.* 2001]. Additionally, assessment data is often scarce, and inadequate to make certain determinations about student knowledge of particular concepts. There are several approaches to solving this problem, including Bayesian networks, other probabilistic means, fuzzy logic, as well as more radical approaches to the modeling problem.

2.5.5(a) Probabilistic Models

In 1987, Tennyson and Park produced the Minnesota Adaptive Instruction System (MAIS) [CoMPIO 2001a], which uses the systems approach to instructional design. The MAIS adapts the topics students encounter using Bayesian predictive statistical models. Question difficulties are adjusted based on student performance. Extensions based on Bayesian approaches are common in the literature today [Millán *et al.* 2000; Murray 1999; Mayo and Mitrovic 2000].

Millán *et al.* [2000] propose a more advanced adaptive approach to minimize intervention by knowledge engineers. Murray [1999] creates an easily implemented linear-time algorithm for Bayesian student modeling that is easily used by ITS developers without deep understanding of Bayesian statistical analysis. Mayo and Mitrovic [2000] use Bayesian networks to predict student performance, going beyond the traditional uses of belief networks in non-predictive student modeling.

2.5.5(b) Non-Probabilistic Models

Alternatives to Bayesian network student models mirror alternatives in AI in general. Fuzzy logic has shown promise in some designs [Vasileva *et al.* 2001], where information theory is used in others [Ueno 2001]. A more radical approach to student modeling removes the stored monolithic knowledge model and replaces it with individual computational structures tailored to each student modeling characteristic [McCalla *et al.*

2000]. This approach, known as active learner modeling, increases the importance of information retrieval in modeling, and possibly signals a shift in the art of knowledge modeling in coming years.

2.5.5(c) Behavior Models

Another approach to student modeling shifts the focus from approximating the level of understanding of each student to understanding the way in which students utilize the tutoring system itself. In Web-based ITS systems that supplement traditional courses, one could argue it is more important for educators to have solid information on the effectiveness of the tools they use, as well as how students make use of these tools. *Branch et al.* [1999] modeled student usage of a video-based multimedia system, using Markov Modeling techniques, and found the data useful for gathering formative and program assessment useful for refining and improving the educational software.

2.5.6 Personalization and Guidance

One strategy towards personalizing tutoring systems to a student's individual needs is to dynamically adapt both hypertext and multimedia. This approach, known as ad*aptive hypermedia* in the literature, has become an active topic of research. Adaptive hypermedia has two goals: *adaptive presentation* and *adaptive navigation support* [Brusilovsky 2000]. Adaptive presentation involves changing the content of Web pages to reflect the individual needs of a user. Adaptive navigation support requires adding, hiding, or ordering the links presented on a Web page to better suit the user's needs.

Adaptive navigation can be used to provide dynamic "guided tours" through the content of a Web site. Beaufils [2000] creates a personalized notepad that allow students to actively construct their own interpretations of the material. Such an approach is consistent with the constructivist paradigm for teaching.

Another constructivist approach to Web-based personalization uses agents in a distance learning setting to tune and integrate resources to individual user's personal traits [Li Xiao *et al.* 2001]. Özdemir and Alpaslan [2000] propose a similar system, using interface agents to guide students through course material on the Web. As one can see,

common uses of intelligent tutoring systems today are to both personalize educational content, and guide students through it.

2.5.7 Interface Design

One area of recent interest to ITS researchers is interface design. Often agent-based intelligent tutors use animated, anthropomorphic representations to interact with the user [Boulay 2000]. These so-called animated-pedagogical agents react emotionally to the actions of their users. Moreno [2001] investigated the role of active participation in anthropomorphic agents, showing that students who learn by participating in a learning task with an agent learn more deeply than students who learn in a non-participating agent-based environment do.

However, support for adding emotion and human-like behavior to ITS interface design is far from universal. Les *et al.* [1999, 14] fear users will attribute more human-like behaviors to the tutor than is justified by the artificial intelligence behind it. While Nijholt [2001] found humans engage in more social behavior with animated pedagogical agents, he questions the emphasis on improving the emotional nature of the agent at the expense of efforts to improve the agents more pragmatic qualities. An animated pedagogical agent was used in an SQL-tutor [Mitrovic and Suraweera 2000] that provides only text-based output, acting as a compromise between simple text-based interfaces and full cartoon-like agents.

2.5.8 Architectural Design

Devedzic [2000] states, "designing the architecture of an Intelligent Tutoring System (ITS) involves a large measure of art." As such, this section does not pretend to offer a taxonomical view of ITS architectural design, but rather a collection of sample approaches to this important task.

With agents being a key metaphor in ITS design, the issue of single-agent versus multi-agent systems becomes important. Badjonski *et al.* [1997] first proposed the use of multi-agent systems in ITS design. They noted that multi-agent systems facilitate easier development by using a "divide and conquer" approach to breaking down complex tasks into several simple entities. Additionally, agent-oriented programming primitives are

easy for programmers to use. In addition, an MAS approach simplifies modification or expansion as well as enables the benefits of mobile agents and distributed computing [1997; Gasser 2001].

Another common architectural theme in ITS design is the use of competing, specialized, differentiated agents to handle learner diversity [Rosselle and Grandbastien 2000; Lelouche 2000b; Heift and Nicholson 2000]. Lelouche [2000b] advocates using collections of agents with specialized functions to handle the issue of learner diversity. Alternatively, Heift and Nicholson [2000] discuss the importance of "generality, modularity and efficiency" in intelligent tutoring systems for handling learning diversity. Roselle and Grandbastien [2000] go further, in proposing a platform for educators to design educational experiments and combine them into a single educational software package. These approaches are popular due to their obvious ability to reduce overall costs of ITS development by encouraging code reuse.

One approach to designing ITS systems is to leverage software patterns, an emerging software technique for identifying and describing reusable, successful solutions to software problems. In reviewing 66 papers on ITS systems, Devedzic [2000] discovers seven ITS patterns. Additionally 63% of all papers supported at least some of these patterns. Of interest was the *application agent* pattern, which describes a layered approach to embedding software agents into existing applications. Several papers discussed here, in addition to this thesis, implement such a pattern. Devedzic [2000] stresses that designers of different ITS architectures use these patterns in most cases without being aware of their existence, and as such *implicit* pattern use is prominent in ITS architectural design.

Other researchers have investigated viewing ITS design from a software engineering perspective. Keeling [1999] proposes a methodology that reduces the involvement of knowledge engineering by separating domain-independent and domain-dependent modules of the tutoring system and tightening the software development cycle. The need for systematic ITS development strategies is also well known [Virvou and Tsiriga 2001]. Hinostroza *et al.* [2000], after observing several ITS projects, have concluded that attempts to dictate student usage models often fail, and that software is used in unintended ways. Patel and Kinshuk [1996] concur, and thus one can gather that

good ITS design involves creating "cognitive tools" that allow individual students to learn each in their own way [Hinostroza *et al.* 2000; Les *et al.* 1999; Paiva *et al.* 1999; Patel and Kinshuk 1996]

Paiva *et al.* [1999] note that most ITS architectures "fail to capitalize on one of the major advantages of such multi-agent systems: the independence and reusability of agents." Most ITS designs result in domain dependence [Galeev *et al.* 2001]. Paiva attempts reusability through the creation of a framework for agents to communicate. This Pedagogical Agents Communication Framework, or PACF, allows for the creation of a heterogeneous community of interacting agents [1999]. Paiva considers three kinds of agents: *tutor agents*, which are responsible for the learning material, *pedagogical interface agents*, responsible for interfacing with the learner, and *domain expert agents*, responsible for the domain knowledge. Paiva's system has proven successful in the creation of learning environments that are reusable across the domains of Astronomy and Math. Galeev *et al.* [2001] describe an as yet untested system for computing the difficulty of learning content in a domain-independent way. This parametric model allows for incorporating student feedback in evaluating the optimal difficulty in learning content.

The lack of a well established developing environment or programming language for ITS and MAS software is a challenge to their adoption by education software developers today [Zarnekow 1998b; Virvou and Tsiriga 2001]. This being so, there are still several attractive programming languages to facilitate MAS / ITS development. Scripting languages like TCL are commonly used in many projects [Zarnekow 1998b], because they allow for rapid development and easy Internet deployment. Traditional object-oriented programming languages, such as C++ and Java, are also popular [1998b]. Java is especially popular because of its support for object serialization, native threading, and Internet and Web compatibility. Other projects use more agent and ITS specific systems such as AGLess [Badjonski *et al.* 1997] and KQML [Paiva *et al.* 1999]. Both of these languages are well suited for agent-development, but lack the programmer base and interoperability of languages such as Java [Zarnekow 1998b].

2.5.9 Artificial Intelligence Paradigms

Several different AI paradigms have proven effective in ITS designs. Many intelligent tutors use expert systems or agents as key components in their systems. Some newer Web-based systems try to combine techniques from several models. A hybrid *expert system* and *neural network* approach has shown promise, combining classical and nouvelle AI approaches [Prentzas *et al.* 2001]. Others have used agents in roles beyond direct pedagogy, including learning "tutoring knowledge" itself [Lelouche 2000a].

A major challenge in ITS design is *planning*, the ordering of content in a curricular flow that matches the student's needs and abilities [Brusilovsky 2000]. One approach to the planning problem is using decision theory. One such system has been designed and tested in the domain of introductory mechanics [Peng-Kiat Pek and Kim-Leng Poh 2000]. Another system that is more ambitious considers student emotions, but lacks a user interface and as of this writing has not been tested with actual students [Murray and VanLehn 2000].

2.5.10 Trends

In reviewing the literature, one finds many examples of sophisticated, intricate ITS systems that provide demonstrable improvements in student performance. In most cases, however, these systems lack broad applicability, due to the domain-dependent nature of their AI systems. Knowledge engineers painstakingly design the expert systems in each to provide custom-tailored models that guide students through the problem solving process. But often such systems make decisions on extremely limited data, and some researchers have questioned whether the sophisticated AI techniques and anthropomorphic interfaces [Boulay 2000; Les *et al.* 1999] really benefit educational software design in the long term. This research is a reaction to such approaches, offering a simpler, broader, domain-independent approach to MAS / ITS designs. The following chapter provides an overview of this design, and places it in its proper context within the literature.

3 Design

The literature is replete with monolithic software designs that provide student resources and tutoring in one unit. This research offers an alternate approach, separating the online resource design and interface from the tutoring design and interface.

While users of PIVoT and the Personal Tutor see an intelligent, unified learning environment, the actual design of PIVoT/PT is rather complex. Several independent modules interact together to provide complex behavior leveraging domain-independent design principles. This chapter discusses the pedagogical and technological approaches to this research, and how these approaches were combined with knowledge drawn from the literature to affect design decisions in the overall PIVoT/PT system. As the design evolved, key components and modules became distinct, and the details of these modules are discussed at length in future chapters.

3.1 Multimedia Pedagogy

As discussed in the literature, the advent of the Web has provided new roles for computers in education. Some educational software developers have tapped the Internet's strength in communication and information delivery, revitalizing interest in distance learning. Others use the Internet as a new means of assessing courseware and students, administering quizzes and exams online [Hoole *et al.* 2002]. Traditional college courses use the Web as a repository for class information, replacing other paper-based (or videotape-based) means of distributing course material [Maddux and Cummings 2000]. PIVoT combines technologies and epistemological philosophies from all of these approaches to multimedia pedagogy, in order to create a supplemental learning

environment that leverages the communication, information delivery, and automation advantages of the Web to supplement traditional courses.

PIVoT plays a supplemental role in the learning process; that is, it provides additional access to alternative versions of course material, often in nontraditional formats. Since it is not intended to be the primary expository source for material in a physics course, PIVoT was designed to emphasize content that can be used to reinforce learning by troubleshooting misconceptions, reviewing key concepts, and exploring material further, in new ways, or in greater depth. For example, PIVoT offers students hands-on, software-based simulations of key physics concepts. One such simulation offers students the opportunity to experiment with projectile motion: varying the initial angle, initial velocity, and wind. Materials such as these allow students to explore concepts already covered in class in new ways that allow students to construct their own mental models of the material, and focus on the topics of greatest difficulty to them. Such a constructivist approach has been supported strongly by the literature [Li Xiao *et al.* 2001; Brusilovsky 2000; Özdemir and Alpaslan 2000].

The Web is an effective delivery tool for information in various forms: video, audio, imagery, and, of course, text. PIVoT, likewise, consists of media of several types. By offering content on the same concepts in several formats, students have the opportunity to use the medium that conveys the material to them the best. Chapter 6 explores student models that automatically detect preferences in learning style and adjust its suggestions accordingly, for use with the Personal Tutor.

As a supplementary educational resource, PIVoT behaves akin to a multimedia library. Like all libraries, multimedia libraries must provide a means of searching for, and retrieving, relevant content. As discussed in the literature review, computer algorithms such as VSM and CSM provide standard, effective means for searching and retrieving electronic documents. Searching multimedia, however, is more challenging. Video, audio, and images lack the digital text that is needed for most algorithms in Information Retrieval. To overcome this, all content must be annotated with text that concisely describes each media item. Such descriptive information is known as *metadata*.

3.2 Metadata

Metadata is defined as "a set of words, phrases or sentences that summarizes and describes what is on a Web site, or on individual pages or sub-sections of that Web site, for the benefit of searching." [Becta 2001] By addressing the "who, what, where and why" of information, metadata allows intelligent agents (both human and software) to use this information in the searching process. As discussed in the previous chapter, several educational metadata standards exist such as IMS [IMS 2001], the Dublin Core [Weibel *et al.* 1998], and SCORM [ADL 2001]. These standards not only assure interoperability between educational systems, but also allow intelligent tutoring systems that work with standard metadata formats compatibility with content not even in existence at the time of its creation and deployment. PIVoT was designed to adhere to the requirements of one such metadata standard, the IMS standard, an extension of the Dublin Core.

The IMS metadata standard, like others, provides a structured framework to describe all instructional media items. While the standards are broad and support many different fields, each with different educational applications, PIVoT focuses on the primary ones that affect information retrieval the most. In particular, metadata in PIVoT focuses on the keywords and "key terms" used to describe each item, the topics to which an item belongs, and information about the relative difficulty of each item. In addition, basic information such as the author, title, and description of each work is also recorded.

As one can imagine, recording metadata manually is time consuming. While there are automatic systems for making an initial estimate at the keywords and topics for some content types[1], most content items require significant manual labor to be annotated properly and fully. In order to expedite and simplify the process, a logging application was developed in Java. This application, the *PIVoT Logger*, allows content experts and their assistants to quickly enter in metadata for all PIVoT-supported media types, including video clips, textbook sections, frequently asked questions, external links to related content, and multiple choice questions. This application simplifies the

[1] There has been much research into detecting key words in a video clip using closed-captioning information.

construction and maintenance of a keyword list, a topic hierarchy, as well as tasks specific to each media type.

3.3 Ontology

All metadata gathered from logging content is stored in a relational database. In ITS design, the educational content information stored in the database is referred to as the *knowledge base* (KB) of the system. The KB is said to ontologically represent the domain. An *ontology* is defined as an explicit formal specification of how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them [Papaterpos *et al.* 2001].

While the literature is replete with projects that design their knowledge bases specifically around the structure of a particular domain, PIVoT takes a different approach. PIVoT's reliance on (and strict adherence to) a domain-independent metadata standard separates it from similar research. This approach has both advantages and disadvantages. PIVoT's ontology provides the key separation between the content and the domain it represents. The metadata used in PIVoT highlights surface relationships: that is, the topology of the particular domain. PIVoT's ontology supports a librarian/assistant metaphor, which directly affected the design of the Personal Tutor as an ITS. The information needed for troubleshooting problem solving skills in-depth (as used in physics domain model tracing tutors) is omitted, since such information tends to be domain specific, and furthermore, not all domains require sophisticated problem solving skills. PIVoT's *domain-independent ontology*, (DIO) is discussed in greater depth in Chapter 4 below.

3.4 Client-Server Architecture

This section begins with a discussion of the three-zone network abstraction model used for PIVoT. It then continues with a review of the client interface, its three-phase usage model, and the choices that guided the design of the user interface. It concludes with a discussion of the limitations of this interface, and how it motivated the need for the Personal Tutor and the Tutorlet API.

3.4.1 Abstraction Model

The PIVoT system is a client-server application distributed across several machines. The PIVoT model has two abstraction barriers between three zones: *client*, *server*, and *backend* (see Figure 2 below). The client, using a standard Web browser and ubiquitous video application, makes requests to three kinds of servers: the PIVoT Web server, the PIVoT video server, and external servers. The PIVoT Web server runs a Java Virtual Machine to support all dynamic page generation: this machine is often referred to hereafter as the WS/JVM. All users of the PIVoT system are required to log in and identify themselves. These connections are secured by using certificates and the HTTPS protocol. The Web server's JVM processes requests from the client, and in turn queries the backend database via JDBC 1.2 (Java Database Connectivity) and the Structured Query Language (SQL).

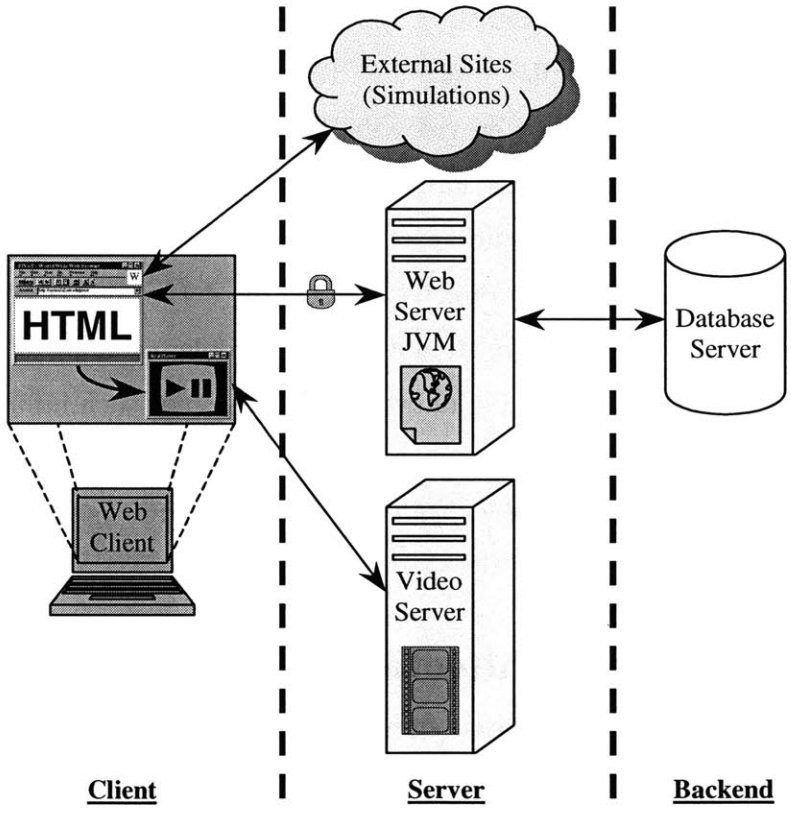

Figure 2: PIVoT Application Network Model

For many content types, the client is redirected to other servers, including some external to the PIVoT-maintained servers. Simulations for example, are located on servers elsewhere on the Internet. The database contains the URLs of these educational

resources, and result pages list these links. To view these, the client clicks on these links, and is then connected to an external site. The client is similarly referred to video content, located on a separate PIVoT-maintained server that speaks a streaming-media protocol. The client receives responses from the server containing URLs that spawn a separate video client, *RealPlayer* (see Figure 2 above). This ubiquitous, multi-platform, freely available client software is used for all video streaming content.

While this discussion refers to the three PIVoT-maintained applications (database, Web server, and video server) as residing on separate machines, this is not necessary. In fact, in PIVoT's current deployment, the Web server and database reside on the same machine. This improves performance with minimal impact on modularity, since the DB, JVM and server applications are isolated and "sand-boxed" by the UNIX operating system on which they all run. The video server (VS) still resides on a separate machine, on a separate MIT subnet, for network performance reasons: the bandwidth and memory needs of the video server are vastly different from the needs of the WS/JVM and DB.

3.4.2 Client Interface

PIVoT's domain-independent back-end for collecting, logging, relating, and storing content from any domain is complemented by a similarly domain-independent front-end. This Web-based interface enables students and faculty to easily search for, browse, navigate, and display instructional content. PIVoT's interface emphasizes redundancy, offering multiple methods to find the same content. Additionally, PIVoT provides software hooks for adding intelligent tutoring modules, allowing the PIVoT interface to easily be expanded in almost any direction.

One key, yet easy, design decision was to use standard Web browsers and HTML text and graphics to interface with the user. Educational software and intelligent tutoring systems before the Web typically ran as stand-alone applications on desktop systems. Since the advent of windowing systems, these applications tended to have a graphical user interface, combining text and graphics. Typically, these interfaces allowed great freedom in presentation format. This benefit is balanced with increased development and deployment cost due to the lack of standards and the need to install the software on each

machine where it will be used. Web-based interfaces allow users to deploy their software cheaply due to the client-server nature of Web development. The majority of software is installed once on a centralized server or farm of servers. Each user of the software need only have installed a Web browser, which is ubiquitous today.

One limitation of standard Web browsers is that they are limited in their ability to display anything but text and images by themselves. *Plugins* are often required to extend the functionality to include video, audio, equations, and other media. Some plugins only run on the more popular desktop platforms (such as Windows), and thus limit how broadly the Web site can be used. Additionally, some HTML features are not equally supported by all browsers, resulting in incompatibilities between different clients. PIVoT was designed to use HTML that is standard across the most popular browsers (Netscape and Internet Explorer), and use plugins only where absolutely needed to render more advanced content. In addition, plugins were chosen for their availability on all popular desktop platforms, including Windows, Mac OS, Linux and UNIX.

One key server design decision was the choice of server platform. Traditional Web servers match each request to a file statically found on the server. PIVoT, like many Web sites available today, needed to produce dynamically generated Web pages based on session state, user preferences, and content stored in databases. Early Web servers provide a solution via an interface for programmers to execute programs with parameters based on the client's request. This Common Gateway Interface (CGI) was powerful but inefficient, since each request required a new process to be spawned. While several server specific solutions[2] to this inefficiency were developed, the Java-based Servlet API provides a platform independent efficient solution that has been adopted by many Web developers.

The developers of the PIVoT project desired an environment that is both easy to develop in and platform independent. A Web server that supports the Servlet API was a natural choice. The Servlet API offers a platform independent computing environment that mimics CGI with support for persistence across client requests through cookies.

Because the Servlet API provides a full Java environment, programmers can leverage easy database connectivity, object-oriented persistence, and a rich existing code base.

The main purpose of the PIVoT interface is to find and retrieve instructional content. This task is decomposed into three distinct phases: *search*, *result*, and *request*. The first phase is typically the search phase, where the user browses through the PIVoT ontology by keyword, topic, or natural language query. Once a search has been entered, or a browse has come to a desired keyword or topic, the result phase is entered. Here, all the content for a particular query is displayed, typically grouped by media type, and sorted in relevance order. If after reviewing the result summaries for each media item the user is unsatisfied with the results, he or she may return to the search phase. If relevant content is found, the user clicks on the link and enters the third phase, where the content is presented. In the third phase video and audio are played, images are displayed, text is rendered, and questions asked or answered. After viewing the content, the user has the option to either return to the result phase to select more instructional content, or return to the first phase to begin a new search.

One challenge in server design was the deployment of the newly digitized video. Each hour-long video needed to be split into well-defined segments for asynchronous delivery to individual users. Commercial software provided part of the solution: the RealVideo server from Progressive Networks, Inc. provides streaming and soft-segmentation support for digital video. Streaming allows a video to be played without first downloading it, and soft-segmentation allows the player to receive and present only the desired portion of a larger clip, without manipulating or duplicating the source. The second part of the solution involved the PIVoT Logger. A video logging module using VCR-like controls utilizing the Java Media Framework enable the Logger to preview each clip and find the precise start and end point of each segment within the source clip, which was recorded alongside the standard metadata kept on each media item: keywords, topics, title, and description.

---

[2] Microsoft and Netscape both created API's for their specific server products, ISAPI and NSAPI, respectively. Both of these APIs, while powerful, are limited to the server platform that created the interface.

3.4.3 Limitations

While the interface provides access to all content in the PIVoT knowledge base, studies have shown that searching through the content can be difficult at times. Before the Personal Tutor was introduced, students often had trouble finding what they needed either by typing in their search or by browsing through keywords or topics. As discussed below in section 6.6.2 and section 7.2.5, many aspects of the interface are not being used to their full or intended potential. While it is clearly possible to redesign the main presentation tools of the PIVoT interface, such changes are often expensive. Redesigning the basic interface may only shift the usability problem to another tool, failing to improve overall usability. Other changes may simplify some tasks, but make others harder and more obscure. In addition, making changes to the entire system interface for all users may help some users and hurt others. It is proposed that a customizable, personalized interface agent is the ideal enhancement to the PIVoT interface. Additionally, instead of creating a single assistant, the agent-to-Web server interface was designed to allow many different agents to be developed cheaply, and separately, from the API that allows their existence. Thus, the Personal Tutor interface was created to extend PIVoT, and the Tutorlet API was created to simplify agent development.

3.5 Tutorlets and the Personal Tutor

This section offers a short overview of the Tutorlet API and the Personal Tutor. Emphasis is given to the history, overarching themes, and principles of its design. Details of the Tutorlet API, such as method names, inheritance trees, and Java-specific issues are omitted here. The full architecture and design of the Personal Tutor and the Tutorlet API are discussed in Chapter 5.

PIVoT, the Personal Tutor and the Tutorlet API are related in a unique architectural way. The Tutorlet API allows for the development of personal tutors. The Personal Tutor deployed and tested with this project is an implementation using the Tutorlet API, but not the Tutorlet API itself. This API was tightly integrated into the standalone PIVoT system, providing hooks through callback and non-callback methods. This allows PIVoT and the Personal Tutor to be separately developed, yet tightly integrated.

3.5.1 Metaphor and Behavior

The Personal Tutor was designed to follow the recently popular agent metaphor. This choice was a natural one, since a tutor follows the major tenets of an agent design: a tutor acts on behalf of its student, the tutor can sense his or her actions, and affect what the student sees, hears, or is otherwise presented. The Personal Tutor aims to do exactly this, being bound to the student that invokes it, and offering suggestions to the user that might be of benefit him or her.

The Personal Tutor was designed so that it could seamlessly integrate with the PIVoT interface. The PIVoT navigation bar and search boxes placed on several pages throughout the site afford the user multiple modes of navigation, each offering a different way to access the content in the PIVoT system. The Personal Tutor aims to provide even further modes for navigating through the educational content. Above the navigation bar present on all PIVoT pages is a switch that allows the user to turn the Personal Tutor on or off. When on, each PIVoT page is vertically expanded slightly at the top to allow for a small four line rectangular area. While on, the PT output window never changes location or size. When the agent wishes to communicate with the user, it may display text or images in this area. The look and feel of the tutor window, in terms of font, color, etc. is made to be consistent with the rest of PIVoT, furthering the illusion that the Personal Tutor and PIVoT are the same. In reality, the Personal Tutor window is governed by an integrated but independent Java-based system, the Tutorlet API.

The primary goal of the Tutorlet API was to create a toolkit for creating domain-independent educational software. This toolkit consists of a collection of algorithms, data structures, and interfaces that enable educational software developers the ability to create complex intelligent behaviors from generic modules and classes. Because each of these modules were designed to be domain-independent, it is easy for developers to create a sophisticated domain-independent tutor from these components. In addition, individual modules can be designed independently of other modules, and combined effectively into a single experiment. This builds on the work of Roselle and Grandbastien [2000], who first proposed such a system.

The Tutorlet API is designed to allow the Personal Tutor to have a life cycle that spans across sessions: each time the student logs in, his or her tutor awakens from its

slumber, informed of the new session, and remembering its state from previous sessions. When the user logs out or chooses another tutor, the tutor is made aware of its impending departure and the tutor automatically records its state so that it may be restored for the next session, whenever that will be.

3.5.2 Agency Model

When designing an agent-based system such as the Personal Tutor, one must determine the *agency model*, or number of agents dedicated to solving the problem at hand. There are strong reasons to represent a tutor as a single-agent system. For example, in the traditional sense, real tutors are a singular entity entrusted with teaching a single student in a one-on-one setting. A multi-agent approach might disorient and confuse a student, with each tutor offering conflicting approaches to a student while he or she is attempting to solve a problem. While this argument is strong for describing traditional tutors, as will later be pointed out, this argument is not as applicable for Web-based tutors.

Due to the arguments mentioned above, the Personal Tutor was designed initially to follow a single-agent model. Each user of the site would be bound to a single tutoring agent, or *Tutorlet*. The name Tutorlet derives from its similarity to – and interaction with – *Servlets*, and from the purported intent of these modules to *tutor*. Each Tutorlet would retain its own state across sessions and have access to information about the user to whom it was bound.

Like all agents, Tutorlets are situated in their environment; that is, they have *sensors* to detect their environment, and *effectors* to interact with it. In the case of Tutorlets, their environment is the PIVoT Web site and the user who is using it. Tutorlets sense the world through both active and passive means. These agents may actively query the user, the PIVoT knowledge base, and the persistent store (described later) on behalf of the user. Additionally, a Tutorlet can passively sense the world through callback methods that notify it when an event has occurred. User events include new page requests, searches, content requests, logins, and logouts.

While a single agent approach appears natural for a tutor, programming one with complex and interesting behaviors is rather difficult. The responsive and reactive nature

of the agent-based systems imply that the behavior of individual agents tend to be simple: complexity emerges from the interaction of these agents [Brooks 1986; Maes 1990]. Since tutors require more complexity than simple information processing applications, single-agent approaches tend to be difficult to program reactively [Jennings and Wooldridge 1998]. A common solution is to take a decentralized, distributed approach, and create a multi-agent system. In such a system, each agent is relatively simple in its behavior, often behaving reactively. The passive callback methods of the Tutorlet API would naturally allow for simple reactive agents, but communication and negotiation between agents would pose a significant challenge.

As the literature shows, negotiation is an active research area in multi-agent systems, with no clear or definitive solutions to the problems that arise. The Tutorlet API allows direct communication with the user by accessing the output stream to place its HTML text in the four-line region near the top of each page. If several Tutorlets were allowed to exist simultaneously, each would have access to this output stream, and there would be no way to ensure that an output limit would be maintained. Furthermore, if the API was modified to maintain a cap on output text, a system to control how the "output rights" would be distributed is necessary to prevent any single agent dominating the resource.

One solution to the above would be to round-robin access to the output stream, per click. Unfortunately, such an approach would prevent agents from communicating with the user on consecutive pages. This is a significant limitation, because the most natural behavior model is for one agent to take control of the session and interact with the user until a task is accomplished. At that time other another agent might take control, and so forth. Communication with a user tends to come in sporadic but continuous bursts, depending on the context of the user in the session.

Understanding the burst nature of communication traffic between the agent and its user, another solution is to allow an agent to take control of the output channel through mutually exclusive control. When an agent wishes to converse with the user, the agent seeks to take the *mutex* shared by all agents, if available. While this allows for continuous bursts of traffic, a key problem with this approach concerns the timeliness of agent communication. The literature has shown a key concern in providing feedback to

students is for it to be presented in a timely manner. If an agent is blocked from the output stream by another agent, the information may no longer be relevant when the resource is freed. For this reason, a significantly different approach was selected to solve the negotiation issue.

The negotiation solution proposed here is a partially centralized, multi-agent model. A new class of agents that lack direct-access to the output stream was created. Rather than communicate with the user directly, these agents, called *Observlets*, communicate with the user through messages called *events* sorted and displayed by a centralized interface agent, similar to the Tutorlet class described above. Each event is assigned a priority by the agent who created it, and the centralized agent delivers them to the user according to availability in the output window. The *Observlet Tutorlet Event* (OTE) model is richer and more detailed than the description given above. Chapter 5 below offers a full of analysis of the OTE model and its component classes.

3.5.3 Pedagogy

In order for the Personal Tutor to provide feedback, it must have data from which to make its decisions. As such, the Personal Tutor is designed to collect data used for assessment purposes. This data is gathered either by asking questions or by observing site usage. The questions used for assessment with the Personal Tutor reflect the pedagogical and technological design philosophies of those who created the system: student control, domain independency, and proven pedagogical strategies.

Given these technological and pedagogical constraints, open-ended questions that required domain-dependent algorithms to model trace the student and to provide feedback were excluded. While the literature shows tutors that take this pedagogical strategy are effective in the domain of physics [Murray and VanLehn 2000], these approaches would not translate to other domains without significant effort from educational software developers with each new domain [Galeev *et al.* 2001]. Instead, self-contained multiple-choice questions were chosen because of their wide applicability to many disparate domains. Additionally, multiple-choice questions have proven their pedagogical value in the literature [Buchanan 2000; Vasileva *et al.* 2001; Hoole *et al.* 2002]. Finally, the self-contained nature of multiple-choice questions allows students to

maintain control over the learning process within each PIVoT/PT session, since students decide when and if to answer online questions. It is important to stress that PIVoT's multiple-choice questions are used by the tutor, but are not used for formal student assessment. While it is possible for these questions to be used in formal assessment, it runs contrary to the nature of PIVoT/PT as a supplemental resource and tutor.

Another key pedagogical feature designed into the Tutorlet API is support for automatic collection of assessment data. Besides providing access to Tutorlets and Observlets of activity from current and previous sessions, a set of tools was developed for modeling and detecting user behavior. These tools will be described in detail in Chapter 6 below.

3.6 Applications

As detailed above, PIVoT is a domain-independent intelligent tutoring system platform for supplementary Web sites. When such a platform exists, the next challenge is using the platform to make a pedagogically useful tutor or tutors. Several technologies were combined or created using the platform to create the Personal Tutor deployed and tested with this research. This section offers several examples of applications of the Personal Tutor. For a detailed look at the implementation of some of these applications, see section 5.7 below. Though many of the examples described here were implemented, time constraints prevented all ideas from being followed through to fruition. These ideas are left for future researchers in domain-independent ITS design.

3.6.1 Context Measurement and Use

The literature has shown the importance of the role *context* plays in understanding a subject. Context, as used here, refers to information about the concepts and topics of interest to a student at a particular point in time. Several classes and modules were developed to track the interests of a user throughout and across sessions. A mathematical model based on Salton's vector space model (VSM) [1975] was used to track and update the context of a session with each successive click. This model was incorporated into an object-oriented hierarchy with tie-ins to PIVoT's query mechanism.

Once context can be easily measured and used to rank content for relevancy, this information can be used by different tutor modules (Observlets) to create content customized for users. Besides ordering search queries according to the contextual relevance of an individual user, contexts may be used to create optimally selected quizzes and occasional suggestions. Such applications are the primary "face" of the Personal Tutor that a student sees.

3.6.2 Messaging System

The Tutorlet API's event model was enhanced to simplify keeping the state of the user across clicks. This system simplifies the tracking of dialogues between users and Observlets, and allows code to be written in a method that follows a logical path. All agents may use the system to post messages that are displayed over several clicks, if space in the output window is available. The agent is notified when a user responds to the message or if it is ignored repeatedly. Based on the response or lack thereof, the agent may act accordingly. This messaging system preserves student-control, allowing users to switch focus between using the PIVoT Web site and the Personal Tutor subsystem. The messaging system allows the developer to focus on the higher-level structure of the tutor's interaction with the user, and avoid the messy state issues in communicating across clicks.

3.6.3 User Modeling API

Another key application of the platform was to create a modeling API, used for learning usage patterns of individual students, groups, and the entire user population. Combined with surveys administered by dedicated "survey observers" via the messaging system described above, users can be categorized by self-proclaimed learning styles. These responses can be used to categorize students and build distinct usage models based on their responses to individual questions. These trained models can be used in future years to predict learning styles and tailor output automatically. This system is described in detail in Chapter 6 below.

3.6.4 User-Constructed Tours

One objective of the PIVoT research project was to simplify educator involvement in the creation of intelligent tutoring content. Toward that end, the PIVoT "tour" system was developed. Tours allow instructors to "record" paths through the PIVoT content that can be traversed by his or her students. Beyond simple linear scripts through content, tours allow the educator to create branches that allow for greater personalization. For example, a tour could lead to a multiple choice question, and offer the user after answering it the option to follow a longer path through extra help, or a direct path to harder material. The Tour Recorder requires no programming skills on behalf of the instructor; users simply browse to the content they wish to add to the tour, and click a button in the Tutor output area.

Additional benefits of the tour system can be achieved by empowering students themselves to create tours. Students can create their own hypermedia notes highlighting relationships between content for their own review of the subject. As such, the Personal Tutor supports constructivist pedagogical philosophies. Additionally, students may share these tours with others. Students may present teachers with content that gives them difficulty, already annotated with comments that assist the instructor in helping the student.

3.6.5 Future Work

One can make greater use of the platform by combining individual modules to create complex pedagogical algorithms. For example, the user modeling API can be used with a simple specific model that measures media type preferences. This model can be trained on each individual user, providing statistics on media type preference. This distribution can be used with the context-ranking module to find not only the most related content, but content in the preferred format. Of course, other future work may explore the application of the platform beyond physics, to observe how effective it is in other subjects. A more detailed description of such future work is discussed below in section 8.1.2.

4 Domain-Independent Ontology

One of the key advancements of PIVoT and the Personal Tutor is their content infrastructure. In order for a tutoring system to provide intelligent feedback, the system must have a well-designed *ontology*, or an understanding of how the concepts of a particular domain are interrelated. As shown in the literature, ontological ITS designs often capitalize on the nature of specific domains. Systems such as Andes use elaborate model-tracing ontologies that leverage specific details about solving physics problems [Gertner and VanLehn 2000]. While it is impossible to create an ontology that allows for pedagogically sophisticated dialogue in all domains, a simple, generic ontology can produce pedagogically adequate understanding by an intelligent tutoring system. In particular, it is argued here an ITS aware of how concepts are related to each other, and what content is related to what concepts, can make intelligent observations about what a student may be interested in and offer predictive suggestions.

This chapter describes the design and implementation of one such system, the Domain-Independent Ontology (DIO) of PIVoT. First, the ontology itself is discussed, identifying the broad classes of entities that make up the DIO. Secondly, this chapter discusses PIVoT's system for *contextual modeling*, which allows the tutor to compute a mathematical representation of the context, or estimated interest, of each user. Thirdly, the *topology*, or interrelationships between the media and attributes of a domain, is discussed. That section reviews innovative algorithms that produce richer information about the pedagogical structure of a domain. Finally, the chapter concludes with a brief discussion of some of the architectural issues involved in implementing such a system.

4.1 Content Infrastructure

The PIVoT DIO consists of three broad *ontological classes*: media, attributes and references. These types, and their subtypes, are organized hierarchically (see Figure 3 below). Each node of this logical hierarchy corresponds to a class in the object-oriented programming sense, allowing for easy implementation in languages such as Java or C++. For reasons already discussed, Java was chosen as the language for implementation of all aspects of PIVoT, including the parts most critical to the Web site and ITS, the server.

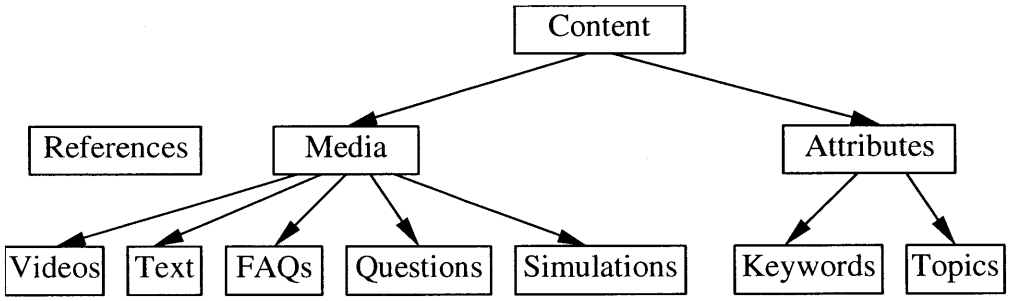

Figure 3: PIVoT's Domain-Independent Ontology

The *MediaItem* class represents each unique educational media resource that could be presented to a user, such as an individual video clip, a textbook section, a practice problem, etc. The *AttributeItem* class represents a descriptor used to describe a media item or link it to concepts or points in the curriculum. Attributes currently consist of either *keywords* or *topics*, representing individual concepts and sections of the curriculum, respectively.

Often it is desirable to conceptually link parts of the ontology to each other, either from media to media, attribute to attribute, or between the two. These links are referred to as references, the final broad class in the PIVoT ontology. Each reference is a *labeled ordered pair*, corresponding to a directed arrow between two content items, labeled with a descriptor. This descriptor indicates the type of relationship the two items have. For example, a book section may be a prerequisite to another book section. The reference would connect the latter to the former with the "has prerequisite" descriptor. Explicit references (created by the logger) are represented by the *Reference* class[3].

[3] There are several kinds of implicit references, which occur naturally in the ontology. See section 4.1.4 on media-attribute relationships and topic-keyword relationships.

4.1.1 Content Identification

The knowledge domain represented in a PIVoT deployment must be stored in a database and accessed by a server capable of executing specialized code that arranges and presents this information. So that both attributes and media can be easily referred to in both the server's Java virtual machine (JVM) and the database (DB), a simple system for retrieving and referring to media and attributes was created. Both media and attributes are unified under the common term, *content*. All subclasses of media and attributes (i.e., FAQs, questions, keywords, topics) have their own identifier *type*. This *type* is a single letter, (in the example given, F, Q, W, and T). Each subclass maintains indexes of all content of its type, assigning each new media item or attribute a unique positive integer, known as the *unique identifier* (UID).

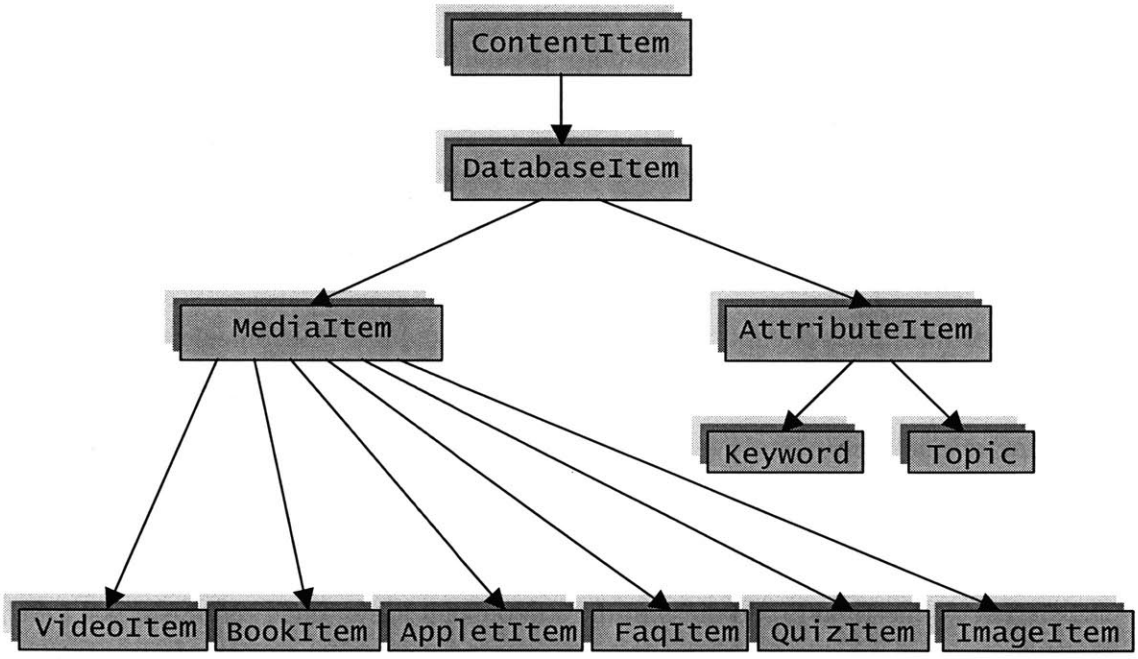

Figure 4: Data Interface Hierarchy

Each unique media item or attribute can be referred to by its content identifier, represented by the parent *ContentItem* class (see Figure 4 above). Each ContentItem consists of a type and UID. For example, a FAQ on projectile motion could be represented by the content identifier F102; the keyword "air drag" could be represented by the content identifier W55, and so forth. Content identifiers are useful throughout the database and virtual machine. For example, references can be thought of as an ordered pair of content identifiers and a label. These short identifiers are useful within not only

the JVM and the DB, but externally, when passing content in text parameters over the network. This is necessary for Servlet parameters passed from Web browser to server.

4.1.2 Media

The PIVoT system represents each educational resource through the *MediaItem* class. As the name suggests, it represents a single item of educational content. The set of all MediaItems in a particular subject form the *media space*. Each MediaItem is atomic and mutually exclusive with all other MediaItems. That is, a textbook represented in the PIVoT ontology would be broken up by section into individual MediaItems, without overlap.

In order for PIVoT to function as a multimedia educational resource, it must store content of several different types. Each media type has its own subclass of MediaItem, and thus has its own set of properties useful in describing itself.

4.1.2(a) Media and Metadata

All media in PIVoT, regardless of type, have certain common properties. These are based upon the IMS metadata standard described above in section 3.2. Though not all metadata types in IMS are applicable to the content types used currently in PIVoT, a core subset of these properties, however, is used for most PIVoT algorithms. Among this subset, of particular interest are:

- *Major keywords* – These words or terms most directly describe the concept or concepts addressed by this content. For example, a textbook section covering "Kinematics of a Rigid Body" would have *angular acceleration*, *angular velocity*, and *moment of inertia* as major keywords.
- *Minor keywords* – These more broadly describe the context of this item within the domain. For example, the same textbook section given above would peripherally involve the concepts of *angular momentum*, *cross product*, and *kinetic energy*.
- *Topic* – This represents the primary location of the problem in a hierarchical course syllabus. Note that this is more focused than the major and minor keywords described above.

- *Difficulty* – It is challenging to define difficulty absolutely and accurately, thus it is represented by an integer between one and five, with five being the hardest. If difficulty is inapplicable or unavailable, zero is used.

4.1.2(b) Property Classification

Some properties are specific to certain media types. For example, it is useful for an electronic version of a textbook section to know the beginning and end page numbers from its corresponding print version. For video segments, the duration of the clip would be a useful property. However, certain key properties, such as title and description, apply to all media types. To implement this, these properties are specified in the MediaItem class, parent to all individual media types. Since it is impossible to create a generic "MediaItem," the class is declared abstract – that is, only subclasses of MediaItem may be instantiated. For most purposes, the algorithms described in this thesis primarily use these common properties, since they allow for the greatest reuse across subjects and educational content types.

4.1.3 Attributes

The most pedagogically useful of the metadata that describes PIVoT content are the keywords and topics. The entire vocabulary of keywords and topics form the *attribute space*. This space is useful in establishing the "location" of content within the domain. This being so, keywords and topics are treated as objects and are a kind of "content" within the DIO (see Figure 3 above), though attributes are not truly content in the educational sense. As such, attributes are more abstract entities than media items.

Each attribute, represented by the *AttributeItem* class, describes a singular concept or topic within a domain. PIVoT's ontology depends on the existence of two distinct subclasses of attributes, *keywords* and *topics*. These are represented by the Java classes *Keyword* and *Topic*, respectively. They correspond to the properties from the IMS standard. When both are used to annotate a media domain, they provide rich descriptive framework that interrelates a given domain in a meaningful and useful way. This system of relationships is discussed below in section 4.3.

4.1.3(a) Keywords

The *Keyword* class is used to describe a single or multi-word term that is key to a specific domain. In the physics domain, examples would include speed, velocity, angular velocity, and linear velocity. Note that while keywords aspire to collectively exhaust the terminology of a particular domain, they are not mutually exclusive. It is possible for two or more keywords to refer to similar or identical topics. For this reason, requiring terms to be mutually exclusive makes the logging task intractable. Similarly, not requiring mutual exclusivity introduces other problems. Thankfully, these problems are smaller and easily resolved.

Firstly, without mutual exclusivity it is possible for multiple terms to be exactly synonymous (such as "corkscrew rule" and "right-hand rule"). Though there are many ways to resolve this, the PIVoT DIO resolves this by establishing a parent-child relationship between a term and its synonyms in the PIVoT database. All keywords have a Boolean property *isKey*. For each group of synonymous terms, one and only one term has a true value for the *isKey* property. All other terms in that group are considered "non-key," and each of these terms contain an internal reference to the "parent term." Whenever a user of the PIVoT logger attempts to annotate media with a "non-key" synonym, the program automatically substitutes the term with its key synonym.

Secondly, it is possible for two or more terms to be *nearly* synonymous (such as "drag" and "air resistance"). Fortunately, this problem may be solved through algorithms that automatically assess logged content for related terms. If several terms tend to be used simultaneously to log content, it can be assumed that these terms are more closely related than terms that do not refer to the same content. Section 4.3.3(d) discusses algorithms for discovering the correlation between keywords.

4.1.3(b) Topics

The other attribute used in PIVoT is the *topic*. Where keywords collectively exhaust the terms in a domain, topics hierarchically arrange the broad concepts of a domain into a syllabus-like structure. The universe of all topics in a domain is referred to as the *topic space*, or more accurately, the *topic tree*. All content has a topic (or rarely, two topics) that fit in this hierarchy, with the entire subject located at the root. Logically,

all subtopics are more specific than their parents, and also parent topics encompass all of their particular subtopics, and their subtopics, recursively. When the siblings of a common parent are ordered properly, a topic tree can be traversed depth first to reveal the syllabus of a course. A sample topic tree (a pruned version of the PIVoT's physics topics tree) is shown below in Figure 5.

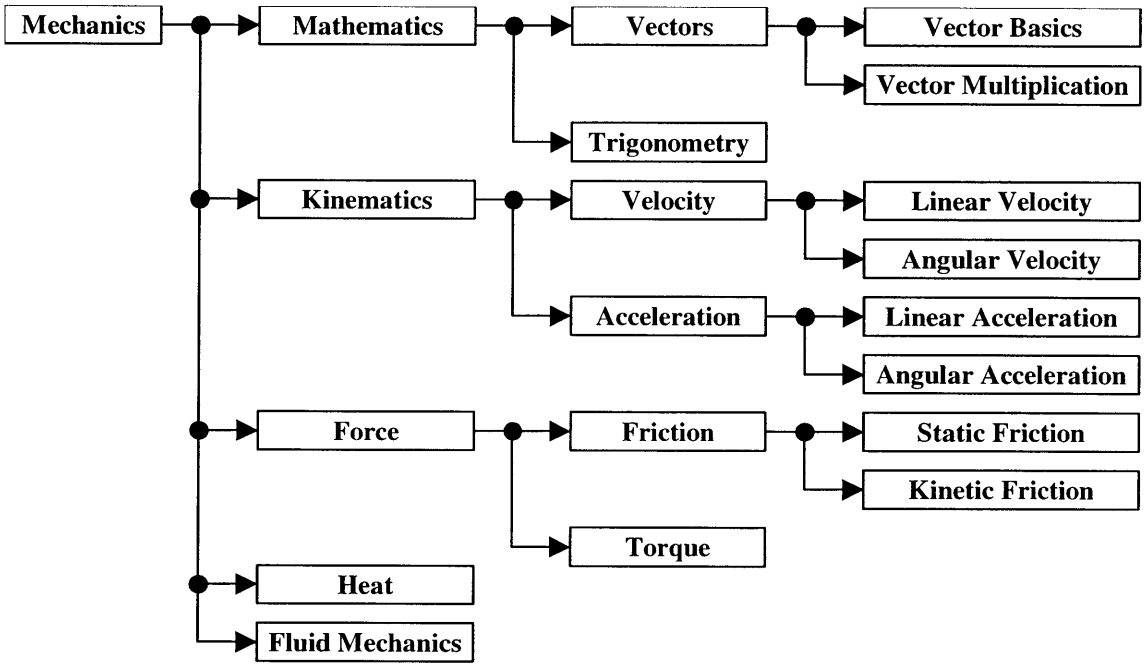

Figure 5: Sample Topic Tree

It is possible for topics and keywords to have the same name, such as the keyword *acceleration* and the topic *Acceleration*. Though at first this seems redundant, the philosophy of the DIO makes the need for the apparent redundancy clear. A keyword is used wherever content refers to a particular concept. A topic is used whenever a particular point in the curriculum is reached. Using the example given above, many media items may have acceleration as a key concept, but very few of these items appear in the course at the point in which acceleration is introduced and discussed. Additionally, not all topics have a keyword of the same name; some topics (such as "Fundamental Physical Constants" and "Fluid Mechanics") are too broad to be a particular concept in the course.

4.1.4 References

PIVoT was designed to enhance the pedagogical nature of multimedia Web sites by making it easy to navigate from content to related content. To partially achieve this, the ontology allows for the manual creation of links between and within media and attributes, called references. There are two kinds of manual references: *manual explicit*, *manual implicit*. Explicit references are created manually with the PIVoT logger, stored in the database, and represented by the *Reference* class. Each of these directed links has a label, known as its *descriptor*, chosen from the IMS vocabulary of pedagogically useful reference types (i.e., "Required By", "Related To", and "Supplement To").

Implicit references are links within the ontology without a corresponding formal entry in the DIO reference table. Currently, implicit references consist exclusively of media-to-attribute keyword and topic links. One can think of each media item's major keyword, minor keyword, or topic as corresponding to an informal, bi-directional, manual link between the media item and the attribute. PIVoT's ontology also supports *automatic* references, generated algorithmically from existing manual links.

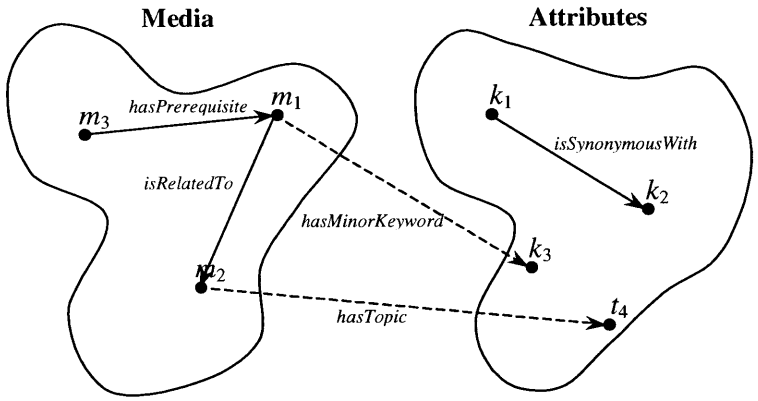

Figure 6: Reference Examples

Automatic references are software-generated links that relate items in the ontology based upon the strength of existing manually entered links. Automatic links may reduce the number of manually entered links necessary to make an ontology useful for intelligent tutoring systems. While the support for these links exist, investigations into algorithms that generate these links is left to future researchers.

While there is no limit on the number of descriptors, it is important to consistently label similar references to make this class more useful to intelligent tutoring systems and

other software that uses this ontology. As discussed above, a small, vocabulary of reference types were chosen from the IMS metadata standard. References shown in Figure 6 are used to show relationships both between and among media and attributes. In this example, k_1 and k_2 are synonyms. Additionally, m_1 is a pre-requisite to m_3. Finally k_3 is a minor keyword to m_1.

4.2 Modeling Context

One of the goals of the tutor is to suggest content that is relevant to the needs and interests of the user. In order to perform this matching task, a mathematical representation of content in terms of its *key features* must be created. Key features may include the keywords, topics, and any other properties that distinguish the content available in the domain. This research project focuses primarily on the use of the attribute space, in particular the keyword space, as the feature space.

4.2.1 Feature Space Dimensionality

This project uses a high-dimensional geometric model of the attribute space to describe points in the domain. Mathematically, each location in the feature space corresponds to a high-dimensional vector. This is based on the vector space model (VSM) [Salton *et al.* 1975] commonly used in Information Retrieval (IR) applications, and discussed in section 2.4 above.

Firstly, all the attributes in a particular domain form the *attribute space* of the domain. The PIVoT ontology can be thought of as having two orthogonal attribute *subspaces*, the *keyword space* and *topic space*. All three models are highly dimensional: each keyword or topic is assigned a dimension in their respective spaces. The design of the DIO makes this assignment easy, since each content item (including attributes) has a type code and corresponding unique ID.

$$v_A = \begin{bmatrix} W1 \\ W2 \\ \vdots \\ W[\dim(W)] \\ T1 \\ T2 \\ \vdots \\ T[\dim(T)] \end{bmatrix}, \dim(A) = \dim(W) + \dim(T)$$

Equation 4.1

Since the algorithms used herein allow the vectors to be sparse but require a known dimension size, a number significantly higher than the upper bound is chosen for the dimensionality of topics and keywords. Although there are 211 topics and 427 keywords in the current PIVoT dataset, a dimensionality of 500 was chosen for the topic space and 1500 for the keyword space. Each keyword or topic can easily be assigned a dimension based on its UID. Equation 4.1 above shows the assignment of dimensions in the attribute space. In this larger space, keywords and topics are stacked into a space with dimensionality of the sum of the respective subspaces (in this example, 2000). Sparse vectors can be implemented efficiently in storage space using hash maps, as discussed below in section 4.4.2.

It is important to note that the assignment of attributes to each dimension result in a high dimensional space. This assignment treats each keyword and topic as orthogonal, where in reality this assumption is untrue: many keywords are synonymous or nearly synonymous, and thus interdependent. Additionally, many topics and keywords strongly overlap. It is possible to reduce the dimensionality by transforming the vector space using matrices that discover the key features of a space, such as singular value decomposition and eigenvector decomposition. Other approaches to correlating attributes are discussed below in section 4.3.3.

4.2.2 Vector Space Model as Context

The vector space model was chosen for the simplicity and power of the cosine similarity metric (CSM). In this model, each document is translated into a vector with all non-negative elements. When two documents are compared for similarity, the cosine of

the angle between their respective vectors (ranging from zero to one, inclusive) measure the degree of similarity between the two. One can create a vector out of a query as well, by having non-zero elements for the key features in the query. In PIVoT, documents are multimedia entities, and as such, the features space is one of the three described above. Due to the natural notion of keywords for describing media, this document will hereafter use the keyword space, unless otherwise specified.

This mathematical model is used to represent both content and queries in terms of their *context*, or position in the feature space. The mathematical representation of context in PIVoT is referred to as a context vector (CV). Two CVs may be compared for relevance, using a function based on the cosine similarity metric (CSM). This function of two vectors is called *co-relevance*, and has a range between zero and one. This section describes the construction and combination of context vectors, as well as the use and properties of the co-relevance function.

4.2.2(a) Computation of Context Vectors

The PIVoT system maps an individual media item to a CV by simply assigning non-zero values to each attribute that describes the content. For convention, major attributes have a value of one, and minor attributes a value of α, a constant between zero and one. This constant is typically one-half or one-third, corresponding to the desired emphasis of major over minor keywords. While there is great freedom in choosing a value for α, this constant is typically chosen by the author based on the average ratio of major to minor keywords per media item. If this ratio is low, a higher α may cause major keywords to be "overwhelmed" mathematically by the minor ones. All non-present attributes, of course, have a zero in the corresponding dimension.

As defined above, the magnitude of a context vector increases with the number of keywords in each MediaItem or query. Since co-relevance measures directional similarity of vectors, the magnitude of either vector argument does affect the value of the function. A context vector is typically formed by linearly combining several vectors. In this case, the magnitude of each of the vectors does affect the direction of the vector, as does the weight of each vector before summation. Normalizing each vector in a sum dilutes vectors with many features in comparison to those with fewer features, since the

magnitude is one regardless of the number of keywords. Weights may then be applied to each normalized vector to manipulate the relative importance of each CV in the sum. Note that if this new vector is linearly combined with another CV, it too may be normalized or weighted. Since the sum of normalized vectors is not equal to the normalization of the sum of vectors, section 4.2.2(c) below offers an algorithm for evolving a context through efficient updates over time that avoids repeated normalization.

4.2.2(b) Computing Co-relevance

The co-relevance of two vectors is simply the dot product of the normalized version of each vector, as shown in Equation 4.2 below[4]. It measures the similarity in direction of two vectors in the same space, since the co-relevance is equal to the cosine of the angle between the two vectors. A co-relevance of one indicates identical direction and the highest relevance. A co-relevance of zero indicates orthogonality, indicating the absence of relevance. Since all vectors in this space have positive elements, a co-relevance of zero means that each dimension has a zero-valued element for one or both vectors: i.e., no keyword or topic overlaps between the two media items.

$$\mathrm{cor}(\vec{v}_1, \vec{v}_2) = \cos(\angle(\vec{v}_1, \vec{v}_2)) = \hat{v}_1 \cdot \hat{v}_2 = \frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1| \cdot |\vec{v}_2|}$$

Equation 4.2

Since a context vector can be created to represent a query identically to the creation of a media item, the co-relevance of a set of media in PIVoT may be compared against a query CV for retrieving what is most relevant. While this evaluation can be performed on the entire universe of content, efficient means of reducing this set have been implemented. It is readily observed that co-relevance will not be positive if there is no overlap between the attributes in the content set and the attributes in the query.

Since the set of media for each keyword is stored in the internal memory structure of the DIO, one can rapidly deduce the smallest set of media for which the co-relevance with the query vector is positive. The *comparison domain* set is the union of the media sets of each keyword. This can be efficiently computed by creating a parse tree where

[4] The notation $\hat{v}$ refers to the normalization of vector $\vec{v}$.

the leaves are the media sets for each keyword, and the inner nodes and root are union operators. By evaluating the parse tree and using a hashing representation for all sets, the comparison domain can be evaluated in expected O(*mn*) time, where *m* is the mean number of media items per set, and *n* is the number of non-zero elements in the CV. Once the comparison domain is computed, the CV for each media item in it is evaluated for co-relevance, and sorted. Those with the highest co-relevance are the content to be presented.

4.2.2(c) Context Vector Example

Consider the following example. Presented here are three normalized context vectors, $\hat{v}_1$, $\hat{v}_2$ and $\hat{v}_3$ shown **Error! Reference source not found.**. In this example, the vocabulary space consists of four keywords, corresponding to each of the four dimensions in the context vector space. The first vector $\hat{v}_1$ represents a combination of keywords 1 and 3, with a 2 to 1 weighting in favor of the first keyword. The second vector $\hat{v}_2$ gives equal weightings to all keywords excluding the keyword 2. The third CV gives significantly greater weight to keyword 2 over keyword 3.

$$\hat{v}_1 = \frac{2\hat{u}_1 + \hat{u}_3}{|2\hat{u}_1 + \hat{u}_3|} = \frac{1}{\sqrt{5}}\begin{bmatrix}2\\0\\1\\0\end{bmatrix} \qquad \hat{v}_2 = \frac{\hat{u}_1 + \hat{u}_3 + \hat{u}_4}{|\hat{u}_1 + \hat{u}_3 + \hat{u}_4|} = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\0\\1\\1\end{bmatrix} \qquad \hat{v}_3 = \frac{3\hat{u}_2 + \hat{u}_3}{|3\hat{u}_2 + \hat{u}_3|} = \frac{1}{\sqrt{10}}\begin{bmatrix}0\\3\\1\\0\end{bmatrix}$$

Equation 4.3 **Equation 4.4** **Equation 4.5**

One notes that the co-relevance of $\hat{v}_1$ and a unit vector for the second keyword is zero, indicating a lack of relevance of keyword 2 to the given context. Additionally, the co-relevance of $\hat{v}_1$ to the first keyword is twice its co-relevance to the third keyword, as we would expect given the specific weightings of $\hat{v}_1$. The co-relevance between vectors $\hat{v}_1$ and $\hat{v}_2$ is high: $\sqrt{15}/5 \approx 0.77$. This is consistent with the high overlap between the two contexts described. The co-relevance of $\hat{v}_1$ and $\hat{v}_3$, however is low: $\sqrt{2}/10 \approx 0.14$. This is consistent with an intuitive notion of relevance, since the two contexts overlap only on keyword 3, and both contexts emphasize their other constituent keywords over the third keyword.

4.2.3 Context and State

Upon understanding the construction, combination, and comparison of context vectors, one can address their use in calculation of state. This section introduces the concept of a *state context vector* (SCV). An SCV represents the "interest" of a user or group of users at a particular point in time. State context vectors are a linear combination of individual context vectors encountered by one or more users during a particular time span. SCVs may be computed for an individual user's session, for the sessions in a given week of the students in a particular class, or the entire lifetime of the course for all students. The weights of each CV are often a function of the time at which it was observed. One such function could be one for all times during a particular session, and zero otherwise. In this case, the state vector would be the un-weighted average of each CV encountered during a session. Other functions bias the average toward activity during a certain time span.

The path of a user through a Web site such as PIVoT is called his or her trajectory. As the trajectory changes, the state of the user evolves. As such, the state context vector is not a static entity — it must continuously be updated according to a set of rules chosen to model the topical interests of a student using the tutor. One observes that recent activity in a user's trajectory is more relevant to current interest than activity in the past. Given this, one approach is to exponentially time-decay the media CVs, giving smaller and smaller coefficients to media viewed further and further into the past. This approach is known as exponential smoothing [Brown and Meyer 1961]. Time can be measured in Web site hits or in elapsed seconds, depending on the notion of context needed for the application.

Consider calculating the context of a trajectory that is n hits in length. Let $t(i)$ representing the time stamp of the i-th hit. Let $V = \{v_1,...,v_n\}$ represent the context CV for each of the n content items encountered along the trajectory. Now one calculates $S(r,V)$, the state CV of the trajectory with a *decay rate* of r. The decay rate is a scalar between zero and one.

$$S(r,V) = \sum_{i=1}^{n} v_i r^{t(n)-t(i)}, 0 \le r \le 1$$

Equation 4.6

When r is equal to one, the time a hit happens does not affect the computation, and thus all CVs are treated equally. In this case, the state context vector is $S(1,V)$, which corresponds to the time-independent average of all locations visited within a site. As r approaches the value zero, the system becomes increasingly "forgetful." In the limit, state information is defined exclusively on the current location of the user within the site. Values in between these two limits correspond to the time-decaying average case, where recent activity is weighed to be exponentially more relevant than information from the past — the lower the value, the more emphasis recent locations are given over past locations.

4.2.4 Simple Global State Algorithm

The formula given above for calculating S, the state context vector requires storing the CVs for the entire trajectory as described. This is potentially costly in a Web site with many users and long trajectories. It is possible, however, to update the SCV incrementally from previous SCVs, as described in the formula below.

$$S_i(r, v_i, S_{i-1}) = S_{i-1} r^{t(i)-t(i-1)} + v_i$$

Equation 4.7

As shown above, a state context vector is updated by weighting the previous SCV by an exponent of the elapsed time, and adding the new vector. This algorithm is constant order in time. Note that the magnitude of each SCV increases with each update. As long as each coefficient is represented by floating point arithmetic, the significant features of the vector remain when normalized, as is done when computing co-relevance.

One might observe that as the trajectory grows, the number of attributes covered increases, and with more non-zero elements in the vector, it loses its sparse nature. While this is so, the values for attributes not recently visited shrink towards zero. One performance improvement for evolving context vectors involves the cutoff of all elements below a certain value ε (epsilon). Every few SCV updates, any value below ε is set to zero. This ensures that the number of non-zero elements does not increase beyond what is necessary.

Note that there is a tradeoff in setting the cutoff constant. Low values of the SCV do affect the suggestions made by the tutor. While several media items may have the

same keywords most recently encountered, past keywords may alter the order of which item to present first. In other words, several clips may both have the most recently traversed keyword: velocity. However, since the user recently traversed media on orbits, these other keywords raise the relevance value for some over others. Setting the cutoff too high will lower the appropriateness of the tutor's suggestions.

Setting the cutoff value too low may increase the size of the state context vectors unnecessarily, and increase the memory and storage requirements of the server needlessly. This extra space and computation time may impact the performance of the server and discourage usage of the tutor.

4.2.5 Evaluation

The usage of context vectors in suggesting content is one of the primary tools for enabling the ITS to model student knowledge and interest. Qualitative, preliminary data from focus groups indicate the effectiveness of this system in providing students with access to content they may not normally search for manually. The evaluation of this system, and others will be discussed further in Chapter 7 below.

4.3 Topology

Most models in intelligent tutoring systems make use of the known relationships in the ontological representation of the course or courses the tutor hopes to teach. PIVoT and Personal Tutor are no different; the knowledge base contains explicit information about the relationships between the content and the syllabus, gathered during the logging process. For example, due to logging, all content is logged for keywords and topics, and these may be used to predict and suggest related content, as discussed above. However, an additional goal of this project is to deduce information about the knowledge base that need not be explicitly defined by content experts. This *implicit* data can be gathered by combining explicit relationships mathematically.

As discussed above, each educational content item is positioned within the domain by three sets of attributes: major keywords, minor keywords, and topics (known as K_M, K_m, and T). There was support and consideration for minor topics in the DIO, but the physics content never utilized this feature. The vast majority of content fits squarely

in a single topic, and thus $|T|$ is rarely two, and rarely higher. While content may not have any minor keywords, all content should be logged with at least one major keyword. There is great informational value in the relationships formed between the content and the union of all attributes that describe them. These relationships form the *topology* of the ontological system. This section discusses various representations of this topology, as well as their applications.

4.3.1 Media-Attribute Graphs

One useful representation of the topology is a graphical one. Consider the content of the ontology to belong to two distinct sets: the media and the attributes. The set of media is called M, and the set of attributes is called A. Connecting these sets are bi-directional arrows, each corresponding to an unordered weighted media-attribute pair[5]. This weight corresponds to the strength of the relationship between the content and its attribute. The set of these media-attribute pairs are referred to as R_{MA}, shorthand for the relationship between the media and their attributes.

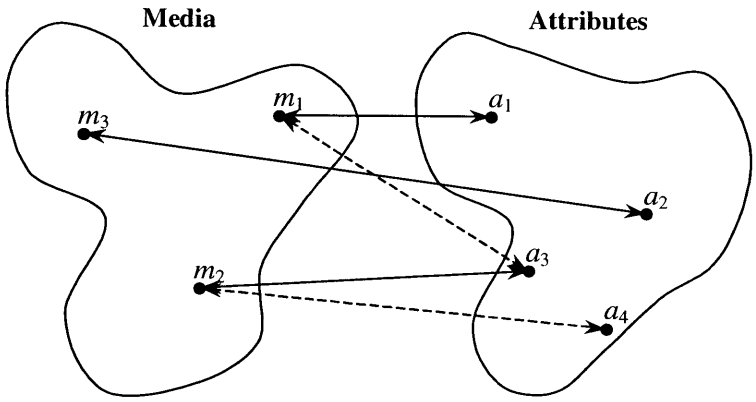

Figure 7: Media and Attribute Space Topology

This view of the topology can be seen in Figure 7 above. Each solid line corresponds to a *major* attribute, and a dashed-line corresponds to a *minor* attribute. The weights have the same numerical values (1 and α) discussed in the previous section. While this graphical interpretation is useful for understanding the nature of the ontology,

[5] These pairs consist of one media item and one attribute only; no pairs consist of two media items or two attributes.

the mathematical representation described below can be used to yield new, implicit relationship between and among media and attributes.

4.3.2 Media-Attribute Matrices

For each R set is a corresponding matrix. This representation is the most useful mathematically. To create this matrix, one must first choose a vector representation for the M and A spaces. First, each individual media item or attribute is assigned a dimension in a media or attribute space. A dimensionality for the M and A space is chosen to be larger enough to accommodate all content.

$$R_{MK} = \begin{bmatrix} 1 & 0 & \alpha & 0 \\ 0 & 0 & 1 & \alpha \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Equation 4.8

The matrix R is a representation of the relationship between the two spaces that correspond to the rows and columns of the matrix. The R_{ij} element corresponds to the relationship between the media item i and the attribute j. As in context vectors, if the relationship is major, the element is one; if the relationship is minor, the element is α, otherwise zero. Since the number of relationships is typically small compared to the number of elements, the matrix tends to be sparse – that is, most of the elements are zero. The example shown in Figure 7 above is converted to an appropriate matrix given by Equation 4.8. Note the dimensionality of this example is vastly lower than what is typical. Nevertheless, the example given captures the essential relationships in a topology.

4.3.3 Correlation Matrices

While keywords and topics are both used to describe the same content, there is no explicit data relating keywords to topics. It is desirable to have a matrix R_{KT} that shows the relationship between keywords and topics. Though no explicitly defined matrix exists, the fact that both attribute types describe the same content can be used to create a matrix that implicitly provides this. This section discusses how to create and use such a matrix.

4.3.3(a) Motivation

Correlating the keywords and topics of a knowledge base can provide useful information to intelligent agents. For example, an agent may wish to know the *vocabulary*, or set of keywords that best describe a topic. Moreover, such an agent may want the relative importance of each of the keywords in the vocabulary. An additional benefit of keyword-topic correlation is calculating the *magnitude* of each of the keywords in the entire ontology: that is, how broadly applicable one keyword is in the entire domain.

4.3.3(b) Design

While there is no direct relationship between keywords and topics, each content-item in PIVoT is separately correlated to both kinds of attributes. In other words, for each content item, there are possibly several major and minor links to keywords, as well as one link (in rare cases, two links) to topics, as shown in Figure 8 below.

When correlating keywords and topics, one must choose the set of content used for correlation. Each content item must be assigned a dimension in "content space" – that is, a positive integer must be assigned for each content item. When a single type is used, one may simply choose the unique identifier (already a positive integer) as the dimension. When multiple types are used for correlation, a scheme can be developed to merge the unique identifiers into a single, non-overlapping span. This can easily be done by assigning a block of the number line to each content type. For example, 1-999 corresponds to B1 to B999, 1001-1999 corresponds to V1 to V999, etc. Since most vectors in the vector space is sparse, there is no problem with leaving parts of the number line unassigned.

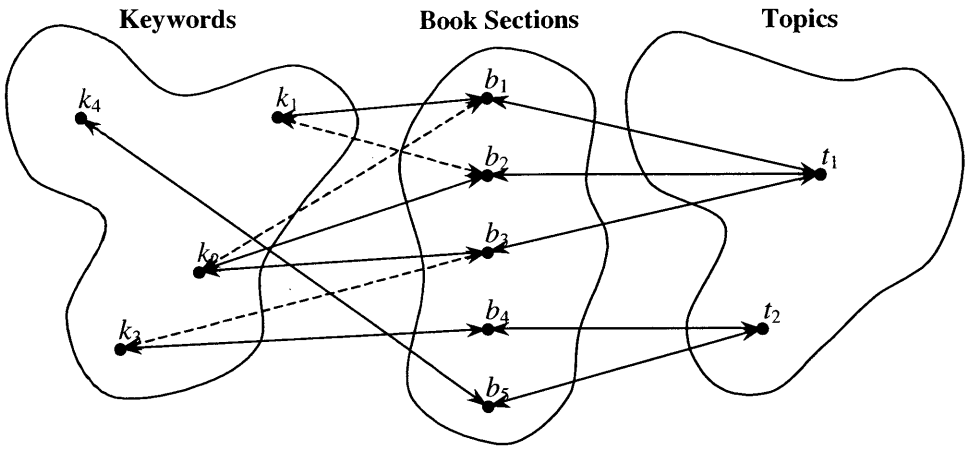

Figure 8: Keyword-Book-Topic Correlation

The example shown in Figure 8 above uses only the textbook sections (referred to as "book" content) as the content space. There is a strong reason for this choice: the entire domain is covered by the textbook at a reasonable level of granularity. Other media types, such as video clips, may have an uneven distribution, with several clips on one topic, and very few clips on another. In a sense, textbook sections form a mutually-exclusive, collectively exhaustive partitioning of the domain. Additionally, besides physics, all other academic domains have textbooks that cover their subject in a similar way.

4.3.3(c) Computation

To compute the correlation matrix, one begins by forming the two *factor matrices* that describe the explicit relationships between keywords and content, and topics and content. Equation 4.9 shows the factor matrix relating book items to keywords, and Equation 4.10 shows the corresponding factor matrix for book items and topics. Note each row in both matrices correspond to a particular content item. Each column in R_{BK} and R_{BT} corresponds to an attribute (keyword for the former, topic for the latter), respectively.

$$R_{BK} = \begin{bmatrix} 1 & \alpha & 0 & 0 \\ \alpha & 1 & 0 & 0 \\ 0 & 1 & \alpha & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 4.9

$$R_{BT} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

Equation 4.10

The *correlation* of a keyword and topic is equal to the dot product of the content space vectors for each. This is based on the cosine similarity metric (CSM) proposed by Salton [1975]. Note that topics and keywords with orthogonal content space vectors have a dot product of zero. Likewise, topics and keywords that are highly related will have large dot products. Due to the nature of matrix multiplication, this correlation computation can easily be computed for an entire domain by transposing one of the factor-matrices and calculating their matrix product[6]. This approach is show in Equation 4.11 below.

$$R_{KBT} = \left(R_{BK}\right)' R_{BT} = \begin{bmatrix} 1+\alpha & 0 \\ 2+\alpha & 0 \\ \alpha & 1 \\ 0 & 1 \end{bmatrix}$$

Equation 4.11

A correlation matrix between keywords and topics serves several functions. Firstly, zero-valued elements indicate that the given keyword and topic are unrelated (or at least, extremely weakly related). Secondly, each column of the R_{KT} matrix, when normalized, yields the relative importance of each keyword in a particular topic. Thirdly, each row of the matrix, when normalized, would yield the relative position of a particular

keyword in the topic hierarchy. Finally, before normalization, the magnitude of each row vector tells something about the importance of the word in the vocabulary: words that appear in lots of content will produce higher magnitudes than those that appear in fewer content items.

4.3.3(d) Applications

Correlation matrices can be used in several ways. They are used in PIVoT to compute the vocabularies for each topic. As shown in the next section, "context" vectors in the keyword space are useful for describing the vocabulary of a subset of the content. For most pages, the context of each page can be described by the keywords of the content on it. When pages corresponding to topics, however, are selected, a system is needed to estimate the keyword vector of the topic. The columns of the R_{KBT} matrix give the optimal keyword space vector for the particular topic. This application is discussed in detail in section 4.2 above.

Another use of correlation matrices is to discover the most important keyword "features" of topic space. These are roughly those with the greatest magnitude. Matrix techniques such as singular value decomposition and eigenvector decomposition may discover the *principal* keywords of a domain. This can be used to segment the domain into smaller sub-domains with little overlap. An intelligent tutoring system could be designed to use a knowledge model based on the relative understanding in each of these sub-domains. Such a system maintains the domain-independent objectives of PIVoT and the Personal Tutor, since all domains could be broken-down in the same way.

$$v_T = R_{KBT} v_K$$
$$w_K = \left(R_{KBT}\right)' v_T$$

Equation 4.12

Another use of correlation matrices is to translate a vector in one space to an appropriately directed vector in another. If a vector represents a student's interest in keyword space, one can discover the appropriate topic space vector by multiplying it by

---

[6] To avoid confusion, A' refers to the transpose of matrix A. The A^T notation is not used since T is used here as a subscript referring to the topic space.

the correlation matrix. The same can be done in reverse, as shown by Equation 4.12 above.

$$R_{KBK} = (R_{BK})' R_{BK} = \begin{bmatrix} 1+\alpha^2 & 2\alpha & 0 & 0 \\ 2\alpha & 2+\alpha^2 & \alpha & 0 \\ 0 & \alpha & 1+\alpha^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 4.13

Finally, another use of correlation matrices is to find the correlation of items within the same space. For example, the R_{KB} matrix may be multiplied by its transpose to produce a matrix that shows the relationship between keywords, as shown above in Equation 4.13. Each value in this symmetric matrix[7] measures how relevant one keyword in the vocabulary is to another. Each row, when normalized can be used to show the strength and presence of related words. This approach may be used with topics as well.

4.4 Architectural Issues

During the implementation of PIVoT, several issues arose involving the efficient implementation of the ontology. Some algorithms and data structures must be carefully designed to make efficient use of time and space. The PIVoT knowledge base resides across several servers, resulting in latency issues that impact the performance of the system. This section discusses the tiered caching strategy that helps alleviate these performance issues.

4.4.1 Server / Database Interaction

The logged information about a domain is stored in a relational database. Queries to the database are made through the standard query language, or SQL. The database may or may not reside on the same physical machine as the Web server. Certain pages require several queries to the ontology, which naively could result in many database queries per Web-server request. For example, the topic tree tool displays an expandable

[7] A matrix B is symmetric if it is square and equal to its transpose: $B = B'$.

tree of the topics in the domain. Rendering the tree requires a depth-first traversal of all the topics in the tree, which could be database intensive with potentially hundreds of DB round trips per request. Though placing both servers on the same machine improves performance by reducing network-induced latency, this cannot always be achieved due to memory constraints. In addition, the placement of both applications on the same machine may result in a lack of modularity. For these reasons, one must take special care to reduce round-trips to and from the database by the Web server.

Besides reducing the number of database queries, another key implementation issue that affects usability involves the transparency of the database in the virtual machine. PIVoT uses an object-oriented class hierarchy to abstract away the details of database access. This approach has several advantages. First, this abstraction removes the intricate details of SQL programming from the higher-level logic of PIVoT and the Personal Tutor. Second, changes to the implementation details, including the brand of database, can be done without changing the application-level code. Finally, similarities between the different content types result in several similar queries being executed. Using an object-oriented layer to abstract database details increase code-reuse, which reduces the possibility for bugs.

Abstracting away the details of database queries, however, impacts the ability to reduce database-server load. Each content item has many properties that can naturally be retrieved by get-methods in the Java API. When a row corresponding to an item is accessed via an SQL query, several columns from the table can be efficiently retrieved together. It should be noted however that not all properties are likely to be needed. When an object is retrieved from the database, not all properties may be used. Those that are used may not be needed on the same hit. If each get-method results in a trip to the database, performance may be reduced. However, if all properties are retrieved in a single query whenever an object is used, space and bandwidth may be significantly wasted.

4.4.1(a) Tiered Caching Strategy

To solve the DB-JVM access problem, a tiered caching approach is used. The most basic properties of all media and attributes are pre-loaded on initialization of the

server, where other fields are retrieved into a volatile cache memory that reduces round trips to the database. Permanently cached properties include the name and unique identifier (UID) for all content, as well as all topological references. The topological information allows each media item to be queried almost instantly for major keywords, minor keywords, and topics. Additionally, each attribute can be queried with reverse-lookup capability, querying for relevant media. Additionally, all topic instances contain permanently cached references to any sub-topics, storing the entire structure of the tree in memory. This allows the Servlets to display topic trees and keyword listings without any DB traffic. Since most of the permanently cached information consists of Java references (pointers) and integers, memory impact is minimal.

Other fields in the database are grouped together into instances of a private inner class *Record* for each object type. When one of these fields is queried by a get-method of the media item's class, the JVM checks the object cache to see if a *Record* instance exists. If not, the *Record* object is constructed, implicitly querying the database and filling all relevant fields simultaneously. Then the media object returns the result of the appropriate get-method of the inner *Record* class.

The cache of these *Record* instances is maintained by a weak hash table [Hayes 1991]. The table is keyed by an integer object corresponding to the UID of the media item. Weak hash tables allow for fast lookup, but allow entries to be reclaimed when the key is no longer in use by the JVM. Care must be taken to insure that the records in memory are allowed to shrink with inactivity. To accomplish this, each get-method places the key UID object for the media item into a queue of fixed maximum size. When the queue is full, the least recently used key is discarded, allowing the record to be destructed and memory reclaimed. Future requests for that object will result in a new record instance being constructed, and consequently a new query to the database.

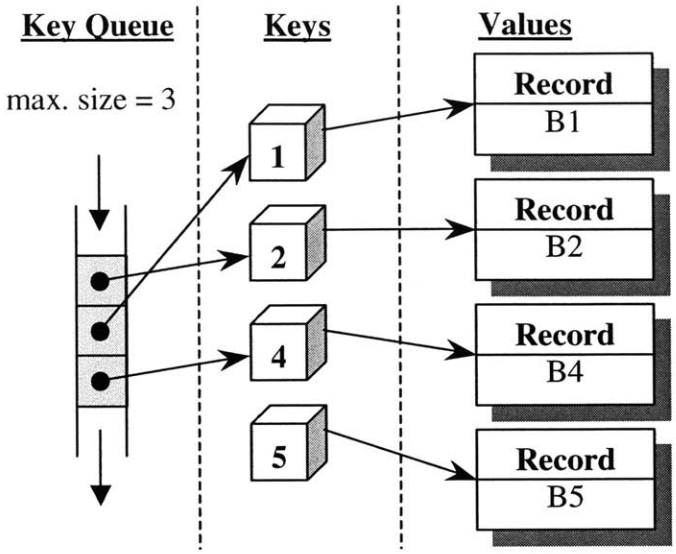

Figure 9: Record Caching System

The record caching system is shown above in Figure 9. Book sections 5, 4, 1, and 2 are accessed by the tutor in that order. After each book section is accessed, `Integer` object keys corresponding to its UID are inserted into the key queue. The key queue has a maximum size of three, so when BookItem 2 is accessed, BookItem 5 is removed from the queue. Each key has a corresponding record value, stored in the weak hash table. Notice that B5 is still in the table, but since its key is no longer referenced by the queue, this record is the only one allowed to be garbage collected from the map. Items B1, B2, and B4 have keys referred to by the queue, and thus are ineligible for garbage collection.

4.4.2 Data Structures

One implementation issue involved the vector and matrix representations used. As is common for VSM models, the vectors tend to be sparse, having mostly zero-valued elements. In other aspects of PIVoT, where Markov models are created, matrices and vectors tend to be dense, having mostly non-zero elements. An abstract Vector and Matrix class was created, with separate concrete classes for sparse and dense implementations of each. Additionally, abstract "row and column" methods of each matrix implementation create anonymous instances of a vector that offer a "vector view" of the source matrix. Other methods iterate over all non-zero values while others automatically remove all values less than a given cutoff ε.

Sparse vectors matrices are easily implemented using a hash map, providing O(1) lookup and O(n) enumeration of all elements (where n is the number of elements). Dense vectors and matrices have array-based implementations. These provide more efficient *get* and *set* performance than the sparse implementations, since no objects are created or removed (corresponding to the index-value mappings) during manipulation.

4.4.3 Serialization and the DIO

All content stored in the database is retrieved as an instance of the DatabaseItem class. This abstract class unifies both media and attribute branches, and extends from the token-like ContentItem class, described above (see Figure 4 above). The DatabaseItem class behaves as an n-ton factory class, preventing other classes from constructing instances of database items. Instead, the class uses public retrieval methods to access instances already constructed by the API. This means that each instance of DatabaseItem is the only instance in virtual memory with that content type and UID. This uniqueness prevents wasteful construction of duplicate objects, as well as duplicate requests to the database for information already stored in memory. The uniqueness of database objects in memory does however pose a problem for object serialization.

Applications written using the Personal Tutor API have the ability to save and remember state from session to session. This is accomplished with Java's serialization API. Java objects, however, do not automatically support serialization, and for good reason. Whenever an object is de-serialized, it becomes a new object, constructed from the serialized form. As such, users can create copies of an existing instance in memory by serializing it and then de-serializing the stored object. In this case both the original and the previously stored version would exist in memory at the same time. Certain PIVoT objects control the construction of objects and prevent copies from being made. For example, database items require that only one instance with a given content identifier may exist in memory at the same time. For this reason, traditional serialization, which copies the fields of an object to storage for later restoration, cannot be allowed to occur.

To overcome this restriction and allow content to be serialized, PIVoT extends the object stream classes for serialization and de-serialization, allowing them to substitute certain instances of objects with a token upon serialization. This token, such as a content

identifier, uniquely identifies the serialized object, but does not contain the object it refers to itself. Upon de-serialization, when a token is retrieved from the stream, it is automatically substituted with the unique instance already in memory, preventing duplicate instances from being created.

To support token-substitution for other classes in the future, a *Lookupable* interface was created. The *Lookupable* interface contains a `getLookupStub()` method that returns a substitute object for the given instance. This object must implement the *LookupStub* interface, which has a `getRealObject()` method that returns the real object to which the stub refers. The MediaItem and AttributeItem classes implement the Lookupable interface, and created their own stub classes to handle the translation.

4.4.4 Expansion

The PIVoT domain-independent ontology was designed to facilitate expansion easily. The logger, database, and Java class hierarchy can easily be expanded to use new types of educational content. After PIVoT's initial deployment, new types were added to the DIO, supporting text documents and still images. These expansions were simple to add to all aspects of the knowledge base, requiring little new code. While the caching structures provide efficient performance for a system whose data is similarly sized to PIVoT, future uses may require more sophisticated caching techniques to achieve the correct balance between performance and capacity.

5 Personal Tutor Design

This chapter offers a more complete discussion of the design of the Personal Tutor and the Tutorlet API, hereafter contracted as PT/API. The basic technological and pedagogical constraints for PT/API were explored in section 3.5 above. Rather than repeat these key points, this chapter continues the earlier discussion in detail, focusing on the technological challenges of merging a multi-agent ITS into an educational Web site. The chapter also offers a review of applications of the Personal Tutor, centering on those implemented and preliminarily tested by a focus group in the spring of 2002.

The Tutorlet API is the chief technological contribution of this research. It offers an alternative to the monolithic approaches to educational software design. Indeed, a key feature of the PIVoT/PT architecture is the ability to run multiple educational experiments at the same time without affecting the main flow of the Web site.

First introduced in section 3.5, the OTE (Observlet Tutorlet Event) model offers a structured way to integrate several small educational experiments into an established Web site. Tutorlets provide centralized control for the Personal Tutor. Observlets, created by the central Tutorlet, or other Observlets, act as observer agents that monitor specific student behaviors. TutorletValue, the persistence interface of the API, allows experiments to communicate with each other across sessions, store Java data structures simply, and record student behavior in a standard way.

5.1 Agents and the Web

It is not coincidental that the rise in popularity of agent-based software occurred after the advent of the World Wide Web. The vast amount of information accessible via the Web necessitates new approaches to assist in navigation. Interface agents, discussed

earlier, offer a friendly "intelligent" interface to assist in navigation of Web sites. Although agent technology seems designed for the Web, embedding agents into Web-based software poses several challenges. This section enumerates these challenges, and how they were met in this research.

5.1.1 Challenges

Maintaining state across individual requests is one of the challenges of Web development. The Web's chief communication protocol, the Hypertext Transfer Protocol (HTTP), is inherently stateless. This being so, without additional measures, each request, regardless of client, is processed by the server independent of all others. This poses a problem to Web designers who wish to create longer communication sequences that go beyond request and response.

Another challenge in designing effective agents for the Web is the limitations in sensing that the client-server relationship of the Web provides. It is difficult in Web-based software to know the actions (and inactions) of the user. For example, it is impossible to know definitively when a user has ended a particular session. While HTTP is a connection-based protocol, each session consists of a sequence of individual requests, with no inter-request "connection" established between client and server. As such, when a user quits or logs out, the server is not notified without the user explicitly pushing a logout button that notifies the client. Similarly, certain client actions, such as hitting the back button in a browser, are not recorded on the server side, further limiting the sensing ability of the agent via the server.

A significant design decision involves choosing the machine that executes the agent's code (see Figure 2 above). Agents may reside in the server or the client. Assuming a Java-based environment (see section 5.1.2 below), server-side agents execute all code in the JVM of the Web Server, via the Servlet API; client-side Java-based agents execute their code in the JVM of the browser, via the Applet API.[8] Placing agents on the client machines seems natural, and distributes the computation, enhancing scalability in the number of clients. Client-side agents, however, are the most distant in networking

[8] Agents implemented in other client/server environments would have similar APIs.

latency terms from the data needed to act. Despite scaling concerns, server-side agents have immediate access to server-specific data and can more readily affect the appearance and layout of each Web page. Additionally, any security needed between the server and the backend could be enforced without divulging this information to the clients.

5.1.2 Solutions

The Java APIs for Web development solve many of the challenges presented here. In fact, the simplicity of Web development in Java, along with its strong object-oriented nature and large code libraries, factored heavily in the choice of Java for this project. The other solutions were developed by the author specifically to simplify domain-independent, multi-agent ITS design. One of the dominant goals of this research was to create a *reusable approach* to ITS design, and not necessarily a particular ITS design itself. Most of this chapter discusses this reusable approach, with the last section dedicated to simple, practical uses of PT/API that serve as an exemplar.

5.1.2(a) Servlet API

The first challenge to be handled was the limitations of traditional Web servers for dynamic, session-oriented, software-generated content. The stateless nature of the Web was partially overcome via use of Java's Servlet API[9]. Servlet-enabled Web servers provide a sophisticated implementation based on the popular but primitive Common Gateway Interface (CGI) discussed above in section 3.4.2. Unlike CGI applications, each client request does not result in the creation of another server process to handle the request. The Servlet API allows several requests to share the same application. Individual requests are handled by callback methods in the *Servlet* class. These classes provide access to a *ServletContext* object, which allows for storage of state, in the form of objects in the JVM that are tied to individual clients. These clients are distinguished via client-side cookies.

---

[9] PIVoT and the Personal Tutor were initially designed around Servlet API 2.0, but were later modified to use version 2.1. Since few advanced features are used, newer versions should work with little or no modifications to the Tutorlet API code.

5.1.2(b) Authorization and Session Management

As discussed above, client-side cookies provide for inter-request state being maintained on the server. The Servlet API provides for easy management of the concept of sessions. Each session begins with the first connection from a new client browser. Sessions are tagged with an ID on the server-side. PIVoT maintains its own security model using the session system. When a session is established, the client is redirected to a login servlet that verifies that the user is one recognized by the backend DB.

As discussed above, it is often impossible to know immediately when a client has completed a session. The Servlet API solves this problem via timeouts, but there still exists the possibility that a session is considered terminated when the user has simply taken a break to read a page or work on a problem. This is mitigated by choosing an appropriate value for the timeout, in this case, 40 minutes. If a user makes a new request after the timeout, the server has already terminated the connection, and must ask the user to login once again, starting a new session.

5.1.2(c) Server-side Agents

Due to the importance of proximity to the server and backend, the Tutorlet API was designed to be a server-side API. The server-side agent approach allows for tight integration with the Web server and the servlets that generate its pages. By placing the agents on the server, they can be notified of, and participate in, all stages of generating a page, with extremely low latency. For example, the life cycle of each request to a servlet involves several software *hooks*[10] where programmers can determine the content-type, headers, and content of each page. Additional callback methods surround the creation and destruction of sessions. These hooks between the PIVoT servlets and the Tutorlet API are described in section 5.1.2(d) below.

[10] A software hook refers to a location in an API where a programmer may add specialized code to handle a user action or system event.

5.1.2(d) PIVoT / PT Interaction

The servlets used in PIVoT serve to search for and present education content stored in the knowledge base in an orderly manner. All PIVoT-specific servlets must provide several functions, chief among them include authenticating users and presenting *boilerplate* text and graphics. Boilerplate information is that which appears on all PIVoT pages, such as a navigation bar, standard headers and footer graphics, and all content from the Personal Tutor. To prevent user disorientation, the literature has emphasized the need for standard information formats, consistent across all pages [Chien-Sing Lee and Singh 2001].

To simplify maintaining consistency, all PIVoT servlets extend from a common *PVServlet* class, which implements the major servlet callbacks, and in turn creates new abstract callbacks that work around the standard PIVoT template. The implementations of the Servlet callback methods use Java's *final* modifier, preventing each subclass of PVServlet from overriding the template code. Instead of overriding these standard callbacks, the abstract PVServlet class provides its own abstract callback methods of a different name. The final PVServlet methods call these abstract methods before and after performing necessary setup code.

For example, the `service()` callback method in the Servlet API is overridden by the PVServlet class to handle each request. This final implementation handles all standard actions for the programmer: it checks to see if a user is logged in (and if not, redirects the user to the Login page), records the request into the database's access log, generates the automatic header, calls the abstract `doPVRequest()` method, and generates the footer. PIVoT programmers implement the `doPVRequest()` method with code that generates the HTML that is unique to that particular servlet.

Another function of the PIVoT implementation of the `service()` method is to handle the display of text for the Personal Tutor. To allow PIVoT and PT developers to work independently, a bridge class was designed to interface between PIVoT and the Tutorlet API. This *TutorletManager* class is bound to each user session, allowing it to be notified by the JVM when a user logs in, or out. The TutorletManager's callback methods are called by the PVServlet instance's final methods. The manager, in turn, calls the appropriate methods of the Tutorlet and all Observlets. The manager also is

responsible for creating, initializing, and serializing all Tutorlets and Observlets. The following section discusses the OTE multi-agent model at the heart of the Tutorlet API.

5.2 OTE Model

This section discusses the design and implementation of the Observlet, Tutorlet and Event (OTE) model. This multi-agent approach simplifies the simultaneous deployment and evaluation of multiple educational experiments, as advocated by Rosselle and Grandbastien [2000]. The OTE model was designed to simplify the illusion of persistence across sessions, offer simple means of negotiation between agents, and allow for easy extension to the model. Layers added on OTE provide for more sophisticated agent-user dialogues, further simplifying the complex task of created Web-based educational software.

5.2.1 Instantiation and Scheduling

The TutorletManager (TM) is the primary controller of the OTE model. As discussed above, it is bound to the servlet session, and thus receives notification from the JVM when sessions begin and end. Additionally, the manager is notified by all PIVoT servlets when requests are initiated by the client, as well as when the Personal Tutor text window needs to be filled. For the output window, the callback passes the output stream as an argument, and the manager is responsible for passing it on to the Tutorlet to use.

The manager is also responsible for initializing all agents (Tutorlets and Observlets) for use by the site. To users of the Tutorlet API, Tutorlets and Observlets appear capable of living (retaining state) across sessions. The TutorletManager is responsible for preserving this illusion by working with persistent database storage (discussed below) to serialize all Tutorlets and Observlets when a user logs out, and restores (de-serializes) them when a user logs in again. When a Tutorlet is run for the first time or an Observlet is created by another agent, the TM is responsible for setting all appropriate values, as would a constructor. Some variables in Tutorlets and Observlets are transient, that is, they are not part of the serialization process. The TM sets these values as well during restoration.

The TutorletManager also serves as a storage and intermediary object for information about the current session, the user of the site, and past sessions of the user (see Figure 10 below). As the diagram shows, the TM also maintains a reference to the Tutorlet as well as a lookup table of all Observlets created for a particular user. The table is keyed by the name of the Observlet. This allows the Tutorlet's other Observlets to look up whether an Observlet is active in the multi-agent environment. This is a key communication mechanism between agents: agents add query methods to their API, and Observlets often select fixed names to facilitate lookup. For example, the `pivot.tutorlet.observlets.ContextObservlet` class, which handles context as discussed in section 4.2 above, chooses its name to match the class's name. There are instances, however, where many Observlets exist simultaneously for a user with the same class name. In these cases, one can either use a numbering scheme, or more typically, *anonymous unique naming*.

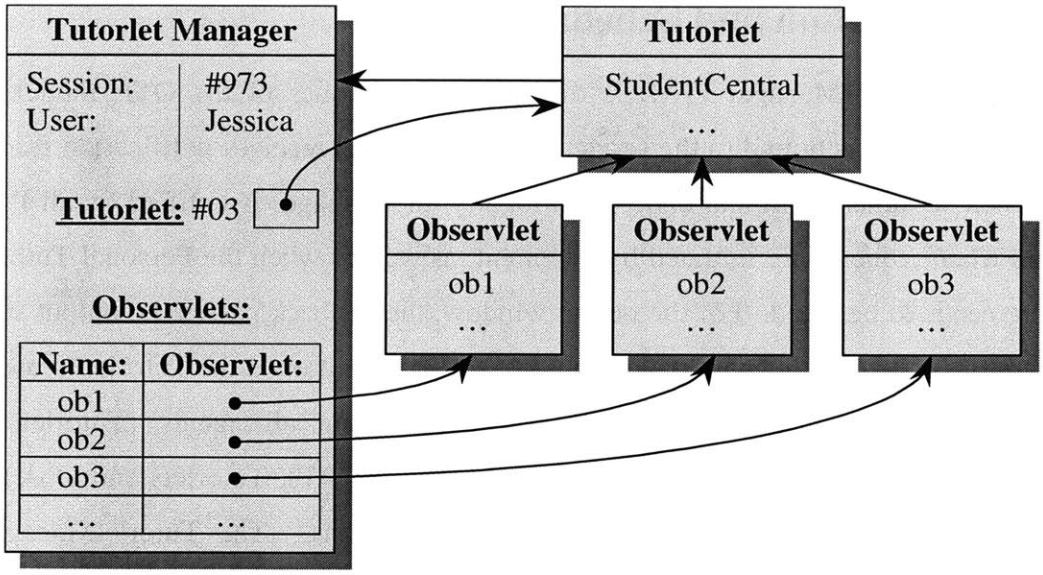

Figure 10: Tutorlet and Observlet Management

Often when the TM creates an Observlet, there is no need for it to be retrieved by other Observlets. Either it does not interact with other Observlets, has a short life span, or both. In these cases, the TutorletManager can assign a unique name for each new anonymous Observlet. The name assigned is returned to the method's caller, but typically is ignored.

5.2.2 OTE Inheritance Hierarchy

The PT/API is not a pure multi-agent system, but rather a hybrid, partially centralized system. The agency model promotes decentralized, multi-agent approaches to tutor design, using several agents with indirect user interaction, called Observlets. Each Observlet receives notification of all user and server actions through callback methods, but must communicate with the user *indirectly* through a singular, centralized *Tutorlet*. While there may be zero or more Observlets active at any one time, there are at most one Tutorlet active per user. When no Observlets are used, PT/API applications act according to a single-agent approach.

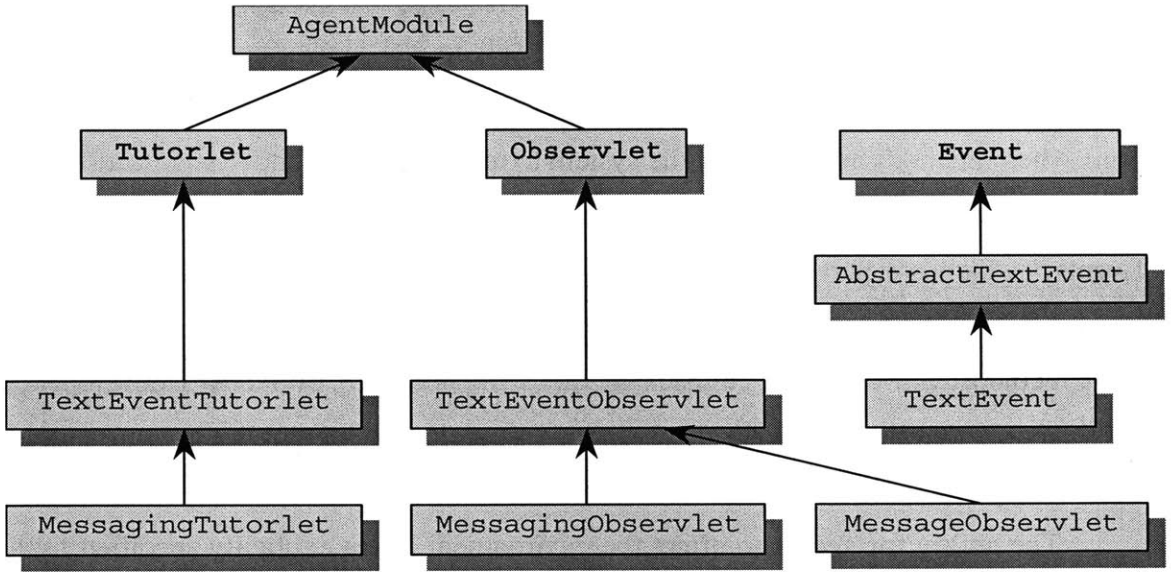

Figure 11: Observlet Tutorlet Event Hierarchy

Tutorlets and Observlets are similar in function, and thus are designed to have the same hooks into user and server actions. To model this similarity, both Tutorlets and Observlets have a common *AgentModule* parent class (see Figure 11 above). All callback methods common to both Tutorlets and Observlets are declared abstract in this base class. For example, since all agents can know the user they serve and add new Observlets, `getUser()` and `addObservlet()` are placed in the base class of the inheritance hierarchy.

Events, used to communicate between the Observlets and their common Tutorlet, have a distinct lineage in the OTE hierarchy (see Figure 11 above). Each increasingly sophisticated event system is built upon the previous layer. Each event model subclasses

the previous layer's Observlet and Tutorlet classes as well. The *TextEvent* layer allows agent modules to pass messages to a centralized Tutorlet who has the ability to handle them in proper order, and display them when appropriate. The *Messaging* layer allows agents to be notified when links in text messages are clicked on, or ignored. These layers are discussed below in sections 5.5 and 5.6.

5.2.3 Agent Interaction

Though the requirements for agency are often disputed [Jennings and Wooldridge 1998; Nwana and Ndumu 1998; Brenner *et al.* 1998], it is universally recognized that agents must interact with their environment. Agents are said to perceive their environment through sensors, and act via effectors. The OTE model allows agents to sense their environment through event-driven callback methods. An event-driven software paradigm is one where methods are called by the system to respond to user and system actions. All such actions are called events. Note that these "event methods" are different from OTE "Events," which are part of a messaging system between agents: Events in the OTE model are self-dated, expiring messages passed between agents via a priority queue. Event methods, however, are callback methods implemented by PT/API developers to add behaviors that handle user and server actions, including logging in, logging out, and other interactions with the server.

The ability for agents to affect the environment is not explicitly specified by the OTE agency model. Clearly, agents affect user behavior through HTML messages placed on the screen in the Personal Tutor output window. These messages may contain active links; when users selecting these links, agents may in turn sense this via callback methods. The method by which multiple agents have access to the output window is only partially specified by OTE. Extensions discussed later in the chapter offer several effective negotiation and communication schemes.

5.2.4 Tutorlets

Tutorlets provide the centralized control in the OTE model. When the client makes a request of the Web server, the TutorletManager passes control of the output stream to the Tutorlet via the `writeResults()` method. Direct access to the output

window is not granted to Observlet agents, and thus this method is not defined in the AgentModule class. The `writeResults()` abstract method must be implemented by all Tutorlets, giving them unlimited, sole control over the Personal Tutor output window.

The Personal Tutor output window has a height of four lines of text. Since the output window is of limited size, programmers must avoid producing HTML that would overflow this area. While it is possible with minor enhancements to the HTML layout of PIVoT to allow the Personal Tutor output to be of larger size (perhaps scrolling the test in a fixed area), the four-line limit was considered by the author to be pedagogically adequate for conveying information without overwhelming the user. As discussed in the first chapter, one objective of the Personal Tutor is to enhance the interface to PIVoT without distracting the user or bombarding him or her with unwanted information. If the tutor wishes to present a substantial amount of information, the student must first click on a link that brings him or her to a page generated by the tutor which contains said information. By forcing developers to be succinct in the suggestions presented to the user, the student retains greater control of the learning process.

While the Tutorlet has sole control of the Personal Tutor output window, the difficulty of designing a single agent that supports sophisticated tutoring functionality forces the Tutorlet to somehow delegate control of the window to other agents. In most cases, the Tutorlet serves to initiate the necessary Observlet modules, marshal their results, and coalesce their results and display them when appropriate. The Text Event model achieves this using the event system of OTE, as described in section 5.2.6(b) below.

5.2.5 Observlets

As stated above, all agents in the OTE, both Tutorlets and Observlets are event-driven. Most agent behavior is a reaction to user actions, such as logging in and out and clicking on links. It was quickly realized by the author that multiple event handlers simplify the development of a tutoring system: Attempts to create single-agent code to handle the complex logic of a sophisticated tutor were mostly fruitless. Individual agents typically have expertise only on a certain kind of user action or behavior. A multi-agent model, involving several "observers," who each contribute to intelligence by

implementing simple, event-driven rules offer an easy way of modularizing complex pedagogical systems. The complexity arises from the coordination and combination of these simple rules, similar to the emergence phenomenon discussed by Brooks [1986] and Maes [1990]. These simple observer agents are named *Observlets.*

Observlets can be instantiated and activated by Tutorlets, as well as other Observlets. This allows for greater task modularity and delegation; whenever a particular user action or behavior needs monitoring, the agent that discovers the need creates and activates a new Observlet, which has the proper code to handle the action. This new Observlet in turn may need to delegate to another Observlet, and so forth. The Observlet class has a public constructor, allowing all to create a new one. Once created, the Observlet must be added via the `addObservlet()` method all agents inherit from the AgentModule class. Observlets may remove other Observlets, or Observlets may remove themselves via the `remove()` method. No security is implemented in controlling which agents may remove each other, thus the OTE model presumes no malicious agents are created and deployed. Considering that Observlets reside and execute on the server, security issues are deferred to the server itself.

The primary design issue in multi-agent architectures is the method of negotiation. Each agent must have some means of communicating with the user and other agents. All agent modules must be notified of each request *exactly once.* Typical callback methods are intended to return quickly to the controller, as in a graphical user interface's event system. For this reason, a single-threaded approach to Observlet notification was taken. The TutorletManager, on each request, contacts each Observlet once in round-robin fashion. All these callback methods may themselves create other Observlets. All newly created Observlets are tracked by the TM, and each of these new Observlets is notified in another round. This process repeats until no new Observlets are created. Of course, care must be taken by programmers to not recursively create Observlets.

Unlike Tutorlets, Observlets cannot directly communicate with the user. This is because each agent has no idea how much of the output window space was used by other agents. Even if output was somehow centrally tracked by each Observlet, the round robin approach to request handling would not guarantee fair use of the limited resource. Some

agents may produce several lines of low priority suggestions, which if scheduled first, could use all of the output area. Lacking space in the output window to post the message, a higher priority alert produced by another agent would be precluded from displaying the message and communicating with the user.

One approach is to limit the amount of output of any single agent on a given request. Another is to prioritize the order in which agents are scheduled. The former approach ignores the practicality that often a single agent has much to say on a particular kind of request, and is silent elsewhere. The latter approach is hindered by the realization that ordering agents is difficult to do in advance of the request. A more realistic solution, advocated here, is to allow access to output only through the posting of "text events." By attaching priorities to these events, one can assure that the limited output area is used effectively.

5.2.6 Events

Events are the third component of the OTE model. Events are messages that can be passed from agent to agent. Typically, events are generated by Observlets and received by Tutorlets, but the OTE model itself does not require this. The *Event* class as defined does not contain any message data in order to support the greatest flexibility in the kind of information an event could contain. Rather, the Event class represents a generic event, and OTE developers are encouraged to extend the class to create a hierarchy of event types. To date, however, all Tutorlets and Observlets use abstract and concrete text events. The current event hierarchy is shown in Figure 11 above. These event types are discussed below in section 5.2.6(b).

5.2.6(a) Event Handling

Events are not necessarily handled in the order in which they are received. All events are assigned a priority represented by the *Priority* class. Using its private constructor, the class creates only seven static constant instances, each corresponding to one of seven discrete priority levels: *lowest*, *lower*, *low*, *normal*, *high*, *higher* and *highest*. By default, events are assigned *normal* priority. Events of higher priority are always handled before events of lower priority, regardless of the order in which the event was

received. Among events of the same priority, however, events are first come, first serve. This allows the structure to behave like a normal queue when most events are of normal priority. If an event needs to be moved to the front of the line, it may be inserted with higher priority, preempting all normal events.

To handle events in this order, a priority queue class was created, extending the heap data structure. Heaps store ordered objects such that both adding new elements and removing the largest element are efficient. Heaps have a performance of O(log n) for the insertion of a new object and O(log n) performance for the removal of the maximum object in the heap, with n being the size of the heap.

Priority queues have an added first in, first out (FIFO) property that is beyond that of a heap. Objects inserted of the same priority must be removed in the order of receipt. This is accomplished by using a wrapper object for values inserted in the heap. This wrapper class is ordered first by the value it encloses, and second by an integer sequence. With each new insertion, the sequence for the new wrapper object is one larger than the previous object. This insures that among objects of the same priority the first wrapper object inserted has the highest value[11]. In order to remove the next value from the queue, the maximal wrapper instance is removed from the heap, the enclosed value is returned, and the wrapper discarded. Since the introduction of the wrapper class does not alter the algorithm for the heap itself, the performance of a heap and a priority queue are identical.

Other structures can also be used for implementing the event queuing system. When there are a small, finite number of distinct priority levels, one can achieve the same prioritized FIFO ordering using parallel FIFO queues for each priority level. Items added to the system are placed in the appropriate internal queue for that priority level. Items are removed by a similar process; each queue is checked for an available item, starting with the queue with the highest priority. This approach is inefficient, however, for systems where the number of distinct priority levels is large, or impossible, where priority has an infinite number of levels, as with mathematical real numbers. To allow for future

[11] The natural ordering of the wrapper class is by ascending priority first, and descending sequence second.

changes to the notion of priority and the event queuing system, the heap data structure was chosen.

5.2.6(b) Event Aging

All events are created with an integer property known as hit-tries. As the name suggests, the hit-tries indicates how many hits an event may remain in the queue before being destroyed. This allows programmers to assess the timeliness of a message at creation. Each event has an age, which naturally begins at zero upon insertion into the priority queue, and is incremented after each processed user request. If an event is not removed from the queue before the age exceeds the hit-tries limit, the message is deemed *stale*, and is removed before it could be de-queued.

The aging of events offers greater control of the presentation of events to the user. Typically, the number of hit-tries and priority are chosen together. A hit-try of one, for example, would force the event to be processed within a single click. This is often accompanied by a high priority to force the event before all others of lower priority. This is done because if the event fails to be dequeued in one hit, it is destroyed. If enough events at the same priority precede the event, then the event is discarded before it becomes irrelevant. In contrast, events that are not time-sensitive, but can be displayed eventually after events that are more important would have a high hit-try number and a low priority.

5.2.7 Extension

The OTE model was designed only to create the framework for developing Web-based, agent-based, domain-independent intelligent tutoring systems. More complete ITS systems are built upon the OTE by extension. As defined, the OTE is too broad as a platform for meaningful ITS development. For example, the text messaging models described below significantly restrict the capabilities of Tutorlets and Observlets by enforcing a more stringent negotiation model. In exchange for these restrictions, one obtains a robust system for rapid development of educational experiments that can compete for student attention.

While the OTE model can be enhanced by extension and restriction of the agent interfaces and negotiation rules, these models depend on key components for maintaining information on students, instructors, and their data. The Personal Tutor API supports the notion of *sections*, or groups, of users, essentially for experiment tracking, aggregation and personalization. In addition, the Personal Tutor supports a sophisticated yet simple persistent storage scheme for sharing information between Observlets, course assessors, and instructors.

5.3 Experiment Control

A recurring theme in the design of the Personal Tutor is the ability to support concurrent experiments. Observlets, as seen below, can compete for the attention of the user via the Messaging model (see section 5.6). Often in educational research, it is desirable to group students into control and experiment groups. The Personal Tutor supports groups via the *TutorletSection* class, and the corresponding database tables. Sections can be thought of as a set of users. More accurately, a section is two sets: a set of instructors, and a set of students. The union of these sets is referred to as the *members* of a given section. The TutorletSection notion of instructor and student is not tied to the registration of the PIVoT user, and thus it is possible for the same user to be an instructor in one section and student in another.

The section system is used primarily to group students and their instructors. Each section has its own permissions as to which tutors the members of the section are allowed to use, and which the members are forbidden to see. Because these permissions can be additive or subtractive, one can create rules that exclude the general population from certain tools, and explicitly override this ban for certain groups of privileged students and instructors. This also allows a programmer to create a new Tutorlet accessible only to the developer. When such a Tutorlet is ready for deployment, he or she may add permissions to the group for which it was intended, or to the user-base at large.

Sections need be neither mutually exclusive nor collectively exhaustive. Sections play a strong role in assigning permissions within PIVoT as well, and certain groups are reserved for administering the site and sections. For example, a so-called "Administrator" group has extra options available on the PIVoT preference page. One

such option is the *Tutorlet Section Manager*, a Web-based user-friendly interface for maintaining groups and permissions. This allows instructors to access PIVoT and update enrollment or enlist students in experimental groups as the term progresses, without intervention from the site developers.

In addition to sections having several members, individual members may belong to several sections. The TutorletSection class provides methods to query the membership of a section, as well as the groups to which a member belongs. It is also desirable for one to immediately know the *primary section* to which a user belongs. For example, each student may be associated with a particular academic section, and in turn, one or more instructors. The TutorletSection class provides a `retrievePrimaryWithMember()` method, which, given a user, returns at most one group. Primary sections are necessarily mutually exclusive; that is, no member may belong to more than one primary section. Primary sections all end in the name "section," so such sections can be created, maintained, and modified by non-technical course administrators.

Primary sections have several uses. By separating students into their corresponding real-world classes, they allow instructors to provide specialized help to their own students. For example, a Tutorlet could be designed to allow instructors to leave personalized messages and content. This was done with the Personal Tutor's Tour system, discussed below in section 5.7.7.

5.4 Persistence

In order for an intelligent tutoring system to be effective, it must have a means of remembering information about the students it hopes to tutor. Tutors work with students via the Web across many sessions, and when students return for further study, the agents must be able to recall key information that will allow for continuity, as well as personalization. In a decentralized model like OTE, this persistent information must also be shared among agents who can build upon the work of other agents. This requires a more organized, systematic storage system, where several agents can agree to a common addressing scheme to insure universal retrieval.

The persistence system is designed as a simple mapping between keys and values. Each key consists of a name and optional user. When the user is omitted, the key is

considered *global*, useful for values not belonging to any particular user, or shared across users. The value is either a simple primitive or Java object.

The persistent store allows names consisting of any string of letters, numbers, underscores or periods. To encourage logical use of the naming space, the author encourages the use of a Java-like "dot" naming system. Tutorlets and Observlets can store their local values using a long name consisting of their class name, followed by a period and the name of the variable they store. The TutorletManager uses this naming system to label the Tutorlets and Observlets themselves, when they are stored or retrieved from the database at the beginning and end of each session. This naming convention prevents naming conflicts between variables with the same name but used by different Observlets. By not making this system mandatory, however, developers have the freedom to use alternate naming schemes appropriate to their application. For example, a developer may wish to organize variables not by class, but rather by pedagogical function.

Values may be of five types: Java objects, integers, text strings, Booleans, and double-precision floating-point numbers. The persistence system is strongly typed by name: when an unused name is bound to a particular object or primitive, no programmer may assign a different type to the same name. This helps insure all agents concur on the type associated with a particular name, reducing software errors.

The persistence system is implemented by a simple Java-based interface and a corresponding database backend. The Java front-end is managed by a single class, named *TutorletValue*. The database consists of seven tables. The first relates names to corresponding unique integer identifiers. The second table relates these integer identifiers to a particular type, represented as an integer from one to five. The remaining five tables, one for each type, relate name and user IDs to their corresponding values. Though wrapper classes exist for treating primitives as Objects, it was decided to have separate tables for common primitive types. This has several advantages: First, storing Java primitives as the corresponding database primitives allow non-Java clients to query the persistent storage for assessment and statistical analysis. Secondly, functions such as summation, maximum and minimum value, and average, can be calculated in the database, passing a single value instead of a larger DB retrieval where the computation

occurs on the Java-side. In addition, the section tables stored in the database can be used to perform these same aggregation functions via SQL over not only all users, but members of particular groups.

To improve performance on the Java end, database query results were cached. The Java cache uses weak hash mapping, maintaining a list of frequently used keys. Both successes and failures are kept in the cache, preventing unnecessary repeat queries for the same key. With each query, the key is added to a MRU (most recently used) queue, with the TutorletValue class ensuring that these values are kept in memory. The maximum size of the MRU queue can be adjusted to match the memory available on the server. In addition, the mapping system conforms to the Java Collection Framework's `Map` interface, allowing future developers the ability to enhance the caching scheme.

The TutorletValue cache is used for read only: all writes are considered "write through," and are sent immediately to the database. To insure data consistency, it is assumed no other database client will write to the store while the virtual machine is running. There is no need for a similar restriction on read-only operations. In addition, it is assumed, naturally, that only one virtual machine is accessing the same database at one time. The ability for other clients to query the store while the system is in use, combined with the transparent storage of primitives, facilitates the jobs of those responsible for formative assessment.

The TutorletValue stores all objects via serialization. For types that cannot or should not be serialized, the TutorletValue class supports the `Lookupable` interface discussed above in section 4.4.3. This allows objects that implement this interface the ability to replace themselves with a suitable stub object that is serializable and is capable of reconstituting the original object, or referring to it, upon deserialization. This process is transparent to users of the persistent store, and is extremely useful when serializing objects from the DIO (see Chapter 4 above).

5.5 Text Event Model

The Text Event model is the first extension to the OTE model. It is built as a layer on top of the *Observlet*, *Tutorlet*, and *Event* classes. The Text Event model was designed as a means of sharing the Personal Tutor output window between all Observlets

and its controlling Tutorlet. All agents produce output by posting HTML messages known as *text events* to a specialized Tutorlet that displays them when room is available. In this model, no agent, including the central Tutorlet, has direct control of the output window.

The Text Event model allows for orderly, fair negotiation between all active agents and the user. Time in the Text Event model is measure in Web requests by the user, also known as a *hit*. All text events are displayed for at most one hit. Like all events, text events have a hit-try number used to determine how many hits can elapse before being discarded without being displayed successfully. Text events ask to reserve a certain number of lines for output. For each hit, the centralized Tutorlet removes all text events in priority order until the output window is filled. Events that are removed from the queue but request more lines than available are placed back into the queue so that another attempt can be made on the next hit. When the window is filled, the queue is empty, or all queued events are larger than the space remaining, the messages are assembled into the output window for the hit.

The original implementation of the Text Event model began by extending the Event class to produce the *TextEvent* class (see Figure 11 above). Text Events have all the properties of Events with the addition of an HTML message and the number of lines to reserve. Since it is impossible for the server to know exactly how many lines a message will require by examining the HTML, the number of lines to request must be given explicitly. If a message requests more lines than is needed, the extra lines are left blank. However, if the message needs more lines than requested, it is possible for the text to overflow the output window.

In order to complete the TextEvent model, both the Tutorlet and Observlet classes need be extended. The *TextEventTutorlet* class extends the Tutorlet class, and overrides the `writeRequest()` method with the final modifier. This prevents all Tutorlet classes that extend from *TextEventTutorlet* from getting control of the output window. Instead, the final implementation in TextEventTutorlet performs the text scheduling and display algorithm described above. The *TextEventObservlet* class is extended with `addTextEvent()` methods that allow for the easy posting of text events. These methods, however, do nothing more than construct a new TextEvent object and pass it on using the

`addEvent()` method in all agents. These same methods are also added to TextEventTutorlet class for completeness.

As defined above, the TextEvent model was incomplete for many applications. First, due to the priority system and competition from other agents, no method exists for knowing when a text event is actually displayed, or destroyed. Second, it is occasionally impossible to choose the message in advance; rather it is desirable to choose the actual message at the point of display, perhaps including information about the page it is to be displayed on. To facilitate this, an *AbstractTextEvent* class was created, that extends from Event and from which TextEvent now extends (see Figure 11 above). Since this class sits in-between the existing classes, it requires no modification to existing applications that already use the TextEvent class.

The AbstractTextEvent class has no message property in its constructors. Rather, the class is declared abstract, requiring classes that extend it to create a `getText()` method. This method passes information about the request it is about to be displayed on as an argument. This allows the class to determine output at the time of display. An additional method is given to retrieve the number of lines the message need reserve. By adding this extension to the AbstractTextEvent, it is possible to greatly increase the flexibility and control over the text events displayed to the user.

5.6 Messaging Model

The Messaging model extends the TextEvent model by further increasing the power of text messages displayed to the user. Messages may contain button-like links that allow users to interact with the tutor. Agents provide methods that handle each possible message response, including the user ignoring the message. Unlike the TextEvent model, these messages may be presented across several pages, allowing the agent to post messages that last longer.

The messaging model is built upon the *MessageObservlet* class, an extension of the TextEventObservlet class (see Figure 11 above). Unlike previous models, messages themselves are assigned their own controller agent, responsible for posting the text events, adjusting the HTML in each event to update links that change as the user moves from page to page, and handle responses if they happen. Often links do not point a user

to a specific page, but rather to the current location, embedding a message to the tutor: MessageObservlets provide a simple way of creating such *dynamic links*. Dynamic linking provides a simple method for designing user-agent interactivity, freeing developers from the tedium of manipulating static links in order to interact with a user across several requests. MessageObservlets also handle presentation difficulties arising from agent competition, using the priority system built into OTE.

Like the TextEvent class, MessageObservlets contain an HTML message, the number of lines needed for display, a hit-try number, and a priority. In addition, these message agents have a *lifespan* property and a *response* method. A message's lifespan measures the number of pages on which the controlling agent must successfully post the message in anticipation of a response. The MessageObservlet class is declared abstract, requiring users to extend the class with its own method that handles any response the user gives by clicking on any of the links in the message.

As discussed above, each MessageObservlet has an abstract `onResponse()` method that contains an integer code corresponding to which button-link a user presses in a given message. For example, an agent may wish to present the text "Do you wish to save? **[Yes] [No] [Cancel].**" The message is displayed for four requests, or until a user selects one of the three links. Each segment of underlined text corresponds to a button-link, written in an augmented form of HTML, described below. When the user does select a link or the message expires, the `onResponse()` callback method is called with a response code argument corresponding to the button chosen. The yes, no, and cancel buttons are assigned positive integers in the order of appearance in the message (1, 2 and 3, respectively). If the message was ignored, the argument 0 is given.

The `onResponse()` method can be used in several ways: the developer may choose to initiate an agent designed to handle another task, ask a follow up question using another MessageObservlet, or perhaps create another MessageObservlet to retry the message if the original communication attempt was ignored. This approach greatly simplifies the handling of logic and dialogue scenarios across multiple requests. The user retains the power, however, to ignore messages and use the Web site without agent assistance.

The links placed within the message property of MessageObservlets differ from links written in standard HTML. First, in order for the controlling agent to be made aware of the response chosen, links must include information about which link was selected, and which question it is in response to. These parameters are added automatically, and may be omitted in the HREF tag of the link. Also, note that in normal Web sites, links point to other pages, and thus must include a URL. With the Messaging model, many links are "dialogue links," serving only to give information to the tutor, and not change which page is being presented. In this case, programmers can supply links without the HREF property (i.e., `<A>[Click Here]</A>`). When the text event is created on the fly by the message's agent, the HREF tag for the current location and appropriate response parameters are automatically inserted. Since the URL of the current page may vary throughout the message's lifespan, these manipulations can be done at display time for each request: this is known as *dynamic* linking. Dynamic linking is accomplished by generating anonymous inner classes that extend from the AbstractTextEvent class. These anonymous classes have sophisticated methods for efficiently inserting and manipulating all links in the message.

By using automatic link generation and manipulation, the Messaging model is significantly more sophisticated than the TextEvent model on which it is based. MessagingObservlet and MessagingTutorlet classes extend from the appropriate TextEvent model classes, providing a similar method named `addMessage()` which automatically creates MessageObservlets similar to how TextEvent objects were added before (see Figure 11 above). One major difference is that messages require a callback method to handle particular responses. Since Java does not provide automatic means for passing methods as arguments,[12] both new classes provide a common abstract handler method that all new MessagingObservlets in turn call. This `onMessageResponse()` method returns with both the identifier of the MessageObservlet that generated the question or message, and the response code for the corresponding messages. While this

[12] Java does have a way of passing a method indirectly, using an interface and anonymous classes that implement said interface. This approach, however, is similar enough to creating anonymous instances of MessageObservlet itself that supporting both approaches was deemed superfluous.

method is useful for quick, simple agents, more sophisticated agents work best by anonymously instantiating a MessageObservlet as an inner-class, thereby including the method in the same place in the code where the agent is created. Such an approach is easier to follow, and allows for sophisticated nesting of agents that handle dialogue.

5.7 Applications and Implementation

Having discussed the technical details and contributions of the Tutorlet and related APIs, this section lists implementation details of applications for the Tutorlet API and Personal Tutor. In particular, this section emphasizes the key modules that comprise the Personal Tutor, as it was most recently deployed for the spring 2002 version of PIVoT. Discussions of the focus group and the deployment history of PIVoT and the Personal Tutor are given below in Chapter 7.

It is important to note that the examples given here are by no means exhaustive; rather, they serve to demonstrate to the reader the power of the system as it exists, and as it could be. The research presented here lays the groundwork for object-oriented, domain-independent intelligent tutoring systems. A discussion of future directions for the Personal Tutor is presented in Chapter 8 below.

5.7.1 StudentCentral

The Personal Tutor was originally designed to produce context-sensitive comments and dialogue based purely on passive detection of user interest and focus. For example, if a user browses to the help page, the tutor would offer the student extra help on the interface, possibly by inferring the difficulty from the recent usage of the user. While this passive, reactive approach is common among agent-based assistants like Microsoft's Office Assistant [Trower 1999], the Personal Tutor supports a more proactive approach to user control of assistant functionality. Using a control panel named "StudentCentral," students have a single menu-driven location to activate and deactivate Personal Tutor features.

The Student Central interface is a simple one-line menu that occupies the top line of the Personal Tutor at almost all times. The menu consists of button-links that either launch PT modules like the Test Your Knowledge quiz generator, or toggle features like

content suggestions on or off. By providing students with an additional menu, the Personal Tutor can act as a supplementary interface to PIVoT. When educational software designers wish to enhance the functionality of an educational supplementary Web site, developers can avoid changing the code of the primary Web, and simply add a menu item to StudentCentral for the new feature. This level of modularity allows the Personal Tutor act as a testing ground for advanced, experimental, or user-customizable features.

5.7.2 MenuObservlet API

The StudentCentral menu is implemented on top of the *MenuObservlet* abstract class, created to simplify the display menus at the top of the Personal Tutor window. The MenuObservlet class, in turn, is built upon the Messaging API, which simplifies tracking responses to menu selections. To ensure the menu appearing at the top of the PT window on all pages, the menu is controlled by a MessagingObservlet with higher than normal priority. The MenuObservlet class provides abstract methods to handle dynamic display of menu buttons, which is necessary to support toggle switches, where the text of a button can change with each click.

The MenuObservlet abstract class also provides a common abstract callback method to handle all responses to menu links. Classes that implement the MenuObservlet offer a consistent, easy to manipulate, user-agent interface. MenuObservlet-based classes such as StudentCentral make powerful use of layered approaches to software design. By layering itself on the Messaging API, which in turn is built on the TextEvent model, and so forth, StudentCentral abstracts complex low-level details into relatively short, high-level code.

5.7.3 Context Observlet

The *ContextObservlet* agent passively observes the trajectory of the user through the site, maintaining global state context vectors (see section 4.2 above) for use by other agents. This agent does not communicate with the user, but rather maintains state information that other agents may retrieve in order to directly communicate with the user. This agent offers methods to allow other agents to retrieve the context for the current

session, the last session, and the entire history of the user. To maintain this information across sessions, the TutorletValue storage system is used.

The sole purpose of ContextObservlet is to update the context of a user based on the formulas discussed in section 4.2 above. As the user moves through the site, the agent converts each page it observes into a position in an attribute space. Pages that present a single media item, such as a book page or video clip, are converted to an appropriate vector in the attribute space. Pages that are the result of a user submitting a search query are described in terms of the attributes used in the search.

Section 4.2 describes three attribute spaces useful in describing media: the keyword subspace, the topic subspace, and the combined attribute space. To simplify design, one common space must be chosen to represent all items. For this application, the keyword space was chosen. This was done since the keyword space can more finely describe each media page than the topic space can, because media items are described by more keywords than topics, on average. While the attribute space would also be an appropriate choice, it is unnecessarily complex since the extra dimensions are often redundant due to the high correlation between the topic and keyword space. Pages that are described by a single topic, such as a search from the topic tree, use the topic to keyword conversion matrix described in section 4.3.3 to create an appropriate vector in keyword space.

The *KeywordContext* class, like its sibling *TopicContext* and *AttributeContext* classes, represents a single context vector (CV) corresponding to one or more media items or attributes. To simplify mapping of content to its corresponding position in attribute space, each class contains methods for constructing vectors given one or more media or attribute items. For conversion of keywords to topic space and vice versa, the appropriate matrix operation is used. Methods also exist to scale and normalize the internal vector representation, as well as a method to cutoff all values less than a given value ε.

The ContextObservlet is designed to compute global state context vectors efficiently, using the techniques described in section 4.2.2(a). At the end of each session, the agent replaces the existing last session CV with the current session CV, and stores both in the persistent store. Other agents, described below, make use of the

ContextObservlet to maintain CV calculations, such as *Suggest*, *Search Assistant*, and *Test Your Knowledge*.

5.7.4 Suggest

The Suggest Observlet is the primary customer of the ContextObservlet agent. This agent uses the context of the entire user's history to produce context-sensitive suggestions to the user. The Suggest feature may be turned on or off using a toggle button in the *StudentCentral* menu of the Personal Tutor. When on, the agent suggests content chosen using the algorithms for efficiently computing co-relevance described in section 4.2.2(b). Each suggestion is posted with lower than normal priority for only a single hit, so that messages that are more important would take precedence, such as menus and user-agent dialogues. To help anthropomorphize the Personal Tutor, suggestions were presented in a conversational manner, including the title and type of the item being suggested. An example suggestion would read, "Based on your recent activity, I recommend this video clip: **[WL Discusses Angular Momentum]**." The underlined and bracketed text represents a link the user may click on that brings the user directly to the suggested item. In this case, it is a video clip of Walter Lewin (WL) discussing angular momentum.

The initial design of the Suggest agent was to suggest only a single content item of a media type complementary to the media the user is currently viewing. That is, when a user is watching a video page, a book section or FAQ is suggested. When a user is reading a textbook section, a video clip or FAQ is suggested. Later, a more sophisticated, personalized approach was taken. The agent was modified to suggest more than one item simultaneously, using media types drawn randomly from a probability distribution based on the preferences of each individual user. This stochastic media preference model is updated by a non-communicative agent similar to ContextObservlet, described in section 6.6 below.

5.7.5 Search Assistant

The Search Assistant agent was created to provide more sophisticated searches than those provided by manually entering each keyword. This Observlet can be activated

through an option on the *StudentCentral* menu. Once selected, a user would be presented with a listing of the top several content items in terms of their similarity to the current user's context.

Like Suggest, the Search Assistant is a consumer of the ContextObservlet agent. Instead of students passively using the few suggestions made at the top of each page, users can actively demand a longer listing of suggestions with the Search Assistant. The Search Assistant provides more relevant results than typing in the individual keywords relevant to a user's interests, since with co-relevance computations, each keyword is weighed according to how frequently and recently the term appears in a user's activity. As such, the Search Assistant can automatically emphasize recently used keywords, while still using older interests to shift the focus of the query.

5.7.6 Test Your Knowledge

The Test Your Knowledge (TYK) feature is the third agent designed to use the ContextObservlet. Like the Search Assistant, Test Your Knowledge presents a list of media items according to their similarity to the current interests of the user. Unlike the Search Assistant, Test Your Knowledge restricts its domain to the multiple-choice practice problems stored in PIVoT. TYK is implemented by, and known internally as, the *QuizGenerator* Observlet.

When the user chooses the Test Your Knowledge menu item on StudentCentral, the student is asked if he or she wishes to view a practice quiz generated from his or her recent usage of PIVoT. Presuming the student does not cancel, he or she is presented with five questions, initially ordered based on their similarity to the user's context. A later version of the QuizGenerator Observlet ordered the questions based on their difficulty. Future research may add more sophisticated means of question ordering.

5.7.7 Tours

One of the most innovative uses of the Tutorlet API is the Tour system, described above in section 3.6.4. With pre-recorded Tours, educators can participate in the content creation and "programming" process. Instructors or students may create a tour using the

Tour Recorder agent, and play it back using the Tour Player agent. Students and instructors in a group may share tours with the *Tour Publisher* feature.

Each Observlet in the Tour recording, playback, and publishing system was implemented using the MenuObservlet base class, vastly simplifying the complex menu-based interactions required with tour manipulation. Complex repeated tasks were also simplified with the creation of abstract Observlets to handle common user-agent interface tasks, such as file-selection, yes/no/cancel dialog boxes, and so forth. Combined, these abstract tools simplify content creation and manipulation.

5.7.7(a) Tour Recorder

While such scripting languages may have been used to create educational content in the past [diSessa 1990], this project combines reusable courseware templates with a web interface and a *graphical* approach to content creation. The Tour Recorder Observlet provides a graphical interface to simplify creation of Tour instances. To begin recording a tour, the user is sent to the PIVoT home page, where all tours begin. The user is allowed to browse naturally through the Web site, and whenever a user discovers a page to annotate with text and links, he or she clicks on the "add page" button. Immediately after, a text box is placed in the Personal Tutor window, where the user may type in a text message, using square brackets to indicate text that should be linked. Once the user submits the on-screen form, the bracketed text automatically becomes linked. Once the user clicks on one these links, the user may browse to the page he or she wishes the link to refer to, and push a "mark page" button. The user is then returned to the source of the link, where he or she can repeat the process for all links on the page.

5.7.7(b) Tour Objects

A serializable *Tour* class was created to facilitate representation and storage of Personal Tutor guided tours. Each Tour instance contains a mapping between unique PIVoT URLs, and the message and links displayed on that page. Due to variances in the order of CGI parameters, or extra parameters added by the messaging API, slightly different URLs may refer to the same content. To circumvent this problem, a system was created to map PIVoT URLs into a unique format that removes details that distinguish

identical pages. For example, all parameters are arranged in alphabetical order, allowing for lexicographic URL comparison. Tours store URLs only in this unique format.

5.7.7(c) Tour Playback

The *Tour Player* agent can be constructed for a given Tour object. Once loaded, the Tour Player presents a message that brings the user to the PIVoT home page. There, the user begins following the tour. The user is presented with the ability to restart or cancel the tour, if one chooses to explore beyond the confines of the tour's path.

5.7.7(d) Tour Publishing

The final component of the Tour system in the Personal Tutor is the *Tour Publisher.* With this module, a student or instructor may post tours for playback by others in the group. This system can be customized for two different pedagogical philosophies. In one approach, only instructors would have permission to publish Tours, following an instructivist view of education. Alternatively, and potentially more powerfully, is a constructivist philosophy, where both students and instructors may publish and share tours, allowing students the ability to construct personally meaningful narrations through interesting or confusing content. This approach allows students to assist each other, transforming a "personal tutor" into a group study tool. The support for both pedagogical philosophies speaks to the flexibility of the Personal Tutor.

5.8 Summary

The Personal Tutor provides several technical contributions to the design and implementation of educational software. The OTE model provides a hybrid agency model that combines the advantages of single- and multi-agent control. OTE allows developers to simplify complex pedagogical objectives into simple, reactive agents that provide for emergent complexity. By remaining neutral to higher-level pedagogical issues, the OTE model allows future research into the effectiveness of different ITS models.

OTE provides only the base layer for educational software design. The Text Event and Messaging models simplify the creation of interactive user-agent dialogue in a competitive multi-agent environment. The TutorletValue persistence module allows

agents to live across sessions, share knowledge, and record information in a manner that allows both developers and course assessors to easily monitor user actions, as well as perform statistics on assessment data.

The Personal Tutor also provides a group permission management system, allowing educators to easily change enrollment in experimental and control group graphically, without programming experience. The innovations described here combine to provide a robust educational software development platform that allows for further research into domain-independent, agent-based, pedagogical software design.

6 Adaptive Learner Behavior Modeling

One of the most promising aspects of the Tutorlet API is its ability to simplify the personalization of intelligent tutoring systems. In order for an intelligent tutor to personalize itself to the user, it must first create a model of the student it wishes to help. Traditional intelligent tutoring systems typically achieve personalization through modeling the knowledge level of the student. While this approach works for most ITS designs where the tutor serves as the primary source of material exposition, such an approach is inadequate for systems where the tutor is part of a supplementary educational resource, as is the case with PIVoT and the Personal Tutor. This chapter describes a stochastic approach to learner modeling that assesses surface-level usage trends instead of conceptual understanding. This approach is well suited for the Personal Tutor, capitalizing on its tight integration with its host Web site, PIVoT.

6.1 Motivation

The supplementary pedagogical role of PIVoT significantly alters the personalization requirements of its integrated tutor PT in comparison to other intelligent tutoring systems. Traditional tutors often serve as a primary source of exposition for course material. As such, these tutors must be effective in correcting conceptual misunderstandings that arise while learning the material, or procedural errors that arise from solving problems in the domain. PIVoT however, is intended for use as a reference to supplement knowledge gained in a classroom or other expository setting. The model proposed in this chapter for the Personal Tutor matches the supplementary role of PIVoT by supporting tasks that enhance the "assistant" role the Personal Tutor plays in student learning. This model can be used to dynamically discover preferences in media type,

search method, and other behavioral patterns. As such, the model provided here is both *behaviorally oriented* and *adaptive*. While it is true that almost all student models attempt to adapt to changes in student interests or behavior, this model's emphasis on predicting and classifying users solely by behavior and application usage stands in sharp contrast to most ITS models that focus on student knowledge.

As stated above, many intelligent tutoring systems focus on modeling the knowledge of a student. Such models attempt to quantitatively represent procedural knowledge, conceptual knowledge, or both. Neither approach is practical, however, with the ontological depth of PIVoT content, as shown below.

Designing an ITS to step a student through the procedural details of solving a problem requires substantial planning and testing, as well as coordination between content-experts in a particular domain and information engineers [Paiva *et al.* 1999; Murray and VanLehn 2000]. In addition, it is almost impossible to design a procedural system to be generalizable enough for all domains. For example, the algebraic mistakes a user may make in a math or physics problem are different from the word-problem modeling mistakes a student might make in a physics or chemistry problem. Even, for sake of argument, if such a domain-independent system were made, the work required to customize such a model for a given domain would make it difficult to use when compared to a domain-dependent design.

Conceptual models attempt to quantitatively measure how well students understand particular concepts in an academic subject, as well as how well they understand the relationships between these concepts. These models typically require a significant amount of assessment data to best understand student performance. As such, conceptual models necessitate strong tutor control of the session's direction. Such high-feedback, strongly directed ITS designs are less effective in supplemental educational Web resources like PIVoT. Data shows students tend to arrive at the site looking for particular information, and are typically unwilling to engage in long sessions. Since PIVoT is not used for formal assessment, students have a choice whether or not to take the multiple-choice tests (and often choose not to). This being so, there is often little data to model student knowledge. Without such feedback, one must focus on other models better suited for use in supplemental Web sites.

The behavioral model described below capitalizes on the lower-feedback, weakly directed nature of PIVoT user sessions. Since students use PIVoT for short sessions with a specific intent already in mind, stochastic models can be designed to quickly deduce preferences in topic and presentation format, leveraging information from past sessions. In addition, multiple-choice surveys can be used to classify students into behavioral groups, based on survey responses. These models can then later be used with probabilistic pattern recognition techniques to detect the pedagogical preferences of each user. The approach used here allows for the creation of multiple stochastic models, each with its own objective in assisting the student.

6.2 Objectives

The primary objective of the Adaptive Learner Behavior Modeling system, or ALBM, is to create stochastic models based on simple, observable, surface-level usage patterns. These models can be used to analyze the behavior of individuals, groups, or the entire user-base. Individual models can be used to personalize the actions and suggestions of the Personal Tutor. Models can be created for groups of students separated by their responses to learning style surveys. These responses can be used in the future to identify learning styles for personalization without student input. Models for all students in a particular classroom section can be interpreted automatically and used to present instructors with information about how his or her students use the educational resource. Models for the entire user-base can be used to provide formative assessment data with minimal human intervention.

The models described here use *observable*, *surface-level* data. That is, ALBM does not attempt to deeply understand the thought processes of each student, nor does it try to understand students' long-term planning objectives. Rather, it focuses primarily on usage patterns that can be observed by the server that measure the path that students take through the site. The usage data can be automatically gathered, analyzed, and repurposed using computational methods in pattern recognition.

As discussed in section 5.1.1 above, not all user actions done via the Web client can be observed by the server. For example, when a student searches for a particular word, and does not find what he or she is looking for, the student may click on the "back"

button on the browser to return to the page that had the search dialog box. Since this action does not contact the server, the adaptive modeling system cannot directly observe this event. It is possible, however, to design adequate models that use only "knowable" behaviors.

As stated above, the ALBM system can be used to assign learning style properties to new students that have not yet been modeled. Probabilistic decision-making can assign ungrouped students to existing groups based on pre-existing models generated from training data from previous terms. As the amount of data available for a particular student grows, or if direct learning style information becomes available, the learning profile of students may change.

Another key objective of this approach to learner modeling is to provide formative assessment data automatically. ALBM models can elucidate patterns in site navigation that hint at the effectiveness of components of PIVoT and the Personal Tutor. This data can be used by instructors and course designers to improve the course and its supplemental Web site.

6.3 Background

The Adaptive Learner Behavior Modeling uses simple but powerful stochastic models to produce its results. There has been significant discussion in the literature on probabilistic behavioral modeling. This section addresses the nature of Markov and hidden Markov Modeling, and how they have been applied to behavior modeling in the past. This section does not give a full review of Markov and hidden Markov modeling, but is rather a short overview of the basic tenets of these stochastic models. For further information on these techniques, refer to the literature in these fields (referenced throughout this chapter).

6.3.1 Markov Models

Markov models are a kind of finite state machine (FSM) with probabilistic state transitions. More specifically, Markov models (MMs) have state transition probabilities dependent only upon the state that one is in within the FSM. In other words, the history of how one reaches a state has no bearing on the probability distribution of transferring

out of that state. This declaration of independence from past states is known as the *Markov assumption.*

The acceptance and applicability of the Markov assumption for a particular problem are essential for valid use of Markov models. It is easy to see how in many applications, past state does indeed have bearing on the choice of current action. In these cases, the problem may need to be modified or another modeling technique chosen. Note that the Markov assumption does not state that Markov models retain no state, but rather that all state information is described solely and completely by the current FSM state. By embedding past history into the definition of each state, one can take a model that depends in a limited way on past state, and convert it into a Markov Model. Such n-step Markov models provide the ability to retain the most recent n steps of history, but do so by exponentially increasing the state space in n.

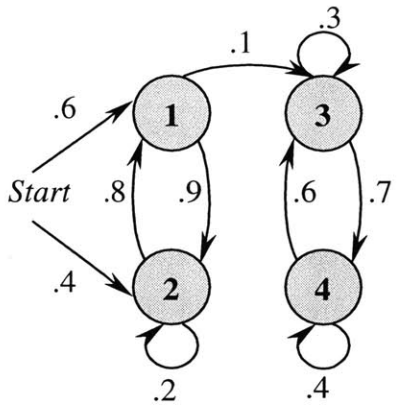

Figure 12: Sample Markov Model

Markov models may be represented graphically or mathematically. Graphically, a finite state diagram is drawn with probability values on the transition edges (see Figure 12 above). Mathematically, an n state Markov Model can be represented by an initial state vector π of length n, and an n by n square transition matrix A (see Equation 6.1 below). The element π_i corresponds to the probability of starting in state i. The element A_{ij} contains the probability of transition to state j when in state i. As such, each row in A and the vector π must sum to one.

$$\pi = \begin{bmatrix} .6 \\ .4 \\ 0 \\ 0 \end{bmatrix} \quad A = \begin{bmatrix} 0 & .9 & .1 & 0 \\ .8 & .2 & 0 & 0 \\ 0 & 0 & .3 & .7 \\ 0 & 0 & .6 & .4 \end{bmatrix}$$

Equation 6.1

Two key probabilistic questions arise regarding Markov models. The first is the easiest: given a sequence of states, what is the likelihood that a given model would generate that sequence? The second is harder: given a sequence of states and an initial MM, how does one optimize the model so that the probability that it generates the sequence is maximized? Both questions are solved by known algorithms in the literature [Rabiner 1989]. These algorithms are discussed in the more general case, using hidden Markov models, described below.

6.3.2 Hidden Markov Models

In some real-world systems, it is impossible to know one's current state with certainty. Often one cannot observe the state of the finite state machine directly, but rather sees the output of such a machine. Hidden Markov models (HMMs) provide a mathematical modeling technique for these systems. Building on the definition of Markov models, each HMM has a set of *observations*, of size *m*. Without loss of generality, these observations are assigned the symbols: o_1, o_2, o_3, ... o_m. At each state, the system outputs a particular observation symbol according to some probability distribution that depends solely on the current state.

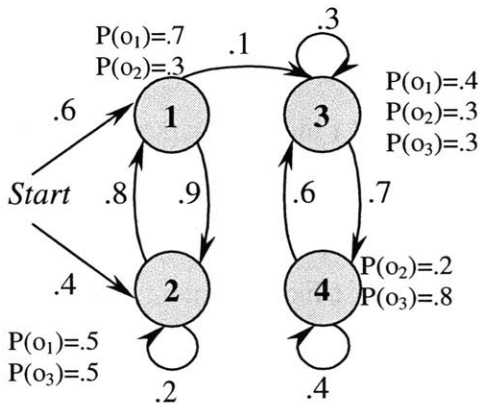

Figure 13: Sample Hidden Markov Model

Like Markov models, hidden Markov models may be represented graphically and mathematically. Graphically, an HMM looks the same as an MM, except for probability statements at each state indicating the non-zero observation probabilities (see Figure 13 above). Mathematically, HMMs add a new n by m observation distribution matrix B to the A and π from its corresponding Markov Model (see Equation 6.2 below). Element B_{ij} contains the probability of observing symbol o_j in state i. As such, each row in the observation matrix corresponds to a probability distribution, and thus must sum to one. An HMM is often referred to by a single $1 + n + m$ by n matrix Λ, which is the horizontal concatenation of π, A, and B. Note all three components must always have the same number of rows.

$$\pi = \begin{bmatrix} .6 \\ .4 \\ 0 \\ 0 \end{bmatrix} \quad A = \begin{bmatrix} 0 & .9 & .1 & 0 \\ .8 & .2 & 0 & 0 \\ 0 & 0 & .3 & .7 \\ 0 & 0 & .6 & .4 \end{bmatrix} \quad B = \begin{bmatrix} .7 & .3 & 0 \\ .5 & 0 & .5 \\ .4 & .3 & .3 \\ 0 & .2 & .8 \end{bmatrix}$$

Equation 6.2

Hidden Markov models are useful in applications where the underlying system is believed to be Markovian, but the details are hidden from view. In essence, such *opaque* models have a "black box" Markov process at their core. It is important to note that all non-hidden Markov models may be thought of as a special case of HMMs. In the MM case, the matrix B is simply the n by n identity matrix I. This case is considered *transparent* since knowing the observation allows one to exactly know what state the model is in. This is in contrast to the imprecise knowledge of state common to hidden Markov models.

With the additional complexity in comparison to Markov models, three key probabilistic questions dominate their use. Two of these questions are similar to the MM case, where one other is unique to hidden Markov modeling. First, given one or more sequences of observations, what is the probability a given HMM generated that sequence or sequences? Second, given a sequence of *observations* and a given model, what is a sequence of *states* that would most likely generate such observations? Finally, given an initial model Λ and a sequence of observations, how does one compute the adjusted model Λ^* that optimally suits the training data?

Rabiner [1989] offers a primer on the algorithmic solutions to these problems. The *Forward-Backward Procedure* (also known as Alpha-Beta) effectively computes the probability of a sequence of observations given a model Λ. The *Viterbi* (or Q*) algorithm provides the optimal sequence of states given a model and sequence of observations. The most challenging problem, training a model to a set of observations, is solved via the Baum-Welch (BW), or Expectation Maximization (EM) method. All these algorithms have been efficiently implemented in Java by the author for use in ALBM. Implementation details are discussed below in section 6.5.

6.3.3 Modeling User Behavior

The literature contains examples of techniques for modeling and analyzing how applications are used. Gorniak and Poole [2000] built dynamic behavioral models from coarse state assignments gathered from analyzing the history of application usage. These states are then refined manually using statistical analysis trying various approaches to split these states into sub-states. These models were Markovian in nature, similar to the models discussed in section 2.5.5(c) above. In the pedagogical domain, Branch *et al.* [1999] also use Markov Modeling to formatively assess the use of a video-based educational tool.

In all examples from the literature cited above, state assignment remains the key challenge. Without a clearly defined understanding of the meaning of the states a user may occupy with an application, the models are meaningless. Additionally, the Markov assumption prevents modeling long-term planning, restricting the nature of conclusions drawn from trained MMs and HMMs.

While the Markov and hidden Markov training and classification algorithms are the same as those found in the literature [Rabiner 1989], the state assignment system and multiple-model design provides a unique contribution to the field of application model. The following section describes the design of the ALBM system, giving special attention to the state assignment system.

6.4 Design

The Adaptive Learner Behavior Modeling system uses standard hidden Markov model algorithms for training and classification purposes. ALBM offers tools for simplifying the creation of modeling the states of a complex, dynamic, Web-based system. This section discusses the key tasks and components of the modeling system.

6.4.1 State Identification

As stated above, the primary challenge in application usage modeling is state and observation assignment. Since the Personal Tutor resides on the server, user modeling is restricted by what is observable by the server. Considering this, the ALBM works closely with the Web server's JVM and recorded logs to provide the most useful models from server-side observations.

The Web-based content presented by PIVoT is dynamic, generated by Java servlets. The only prominent static content is the online textbook; even these pages have dynamic content added, such as standard boilerplate headers and footers. This fact presents a challenge in modeling usage – the number of unique URLs is virtually uncountable. To solve this problem, a method must be devised to categorize URLs by common objectives. Fortunately, the PIVoT system already had a careful and exhaustive logging system in place that writes each record as it happens to the database. Each servlet and static page has an associated unique identifier. Additionally, each piece of content viewed, such as book pages and video clips, have their own unique identifiers. Each HTTP request to the server is assigned a unique **request ID**, which is used to cross-reference other logged entries in the database. Additionally, each user session, has its own identifier, which can be cross-referenced against the user that initiated the session.

Each session can be viewed as a finite sequence of observations, terminating when the session is ended, either explicitly or by timeout. Sessions can also end if the web browser is closed and restarted, as the client only remembers session information for the life of the browser. While it is possible to connect such sessions together if the end of one and the beginning of another falls within a certain continuity time (for example, five minutes), this was not done for the purpose of the analyses done here.

6.4.2 Observation Mapping

Choosing the vocabulary to describe all observations was a major design decision in this project. Too small a vocabulary would hide detail about how the system is used. Having too large a vocabulary would likely increase the number of model states and increase unreasonably the computation time for working with these models. Additionally, increased state granularity might obscure aggregate trends in PIVoT/PT usage.

6.4.2(a) Basic Vocabulary

Initial experiments used a moderate, basic vocabulary, consisting of 20 two-letter symbols. Similar actions were aggregated together to limit the size of the vocabulary. These symbols were chosen to represent the key objectives of the PIVoT user, shown below in Table 1.

Table 1: Basic Observation Vocabulary

Symbol	Description	Symbol	Description
HP	Home page presented	SP	Search page presented
FP	FAQ presented	SR	Search result
BI	Book index	QP	Quiz presented
BP	Book page presented	QA	Quiz answered
TP	Topic Tool presented	JP	Java simulation presented
TB	Topic Tool browsing	VP	Video page presented
TR	Topic Tool result	RV	Real Video Player spawned
KP	Keyword Tool presented	LP	Lectures page presented
KB	Keyword Tool browsing	LV	Lecture Video spawned
KR	Keyword Tool result	LO	Logged out, end of session

Some symbols, such as **BP** (any book page), represent several possible URLs — providing a useful level of aggregation for this study. Some symbols deal with the different stages of the usage cycle. **TP**, **KP** and **SP** represent the initial presentation of a particular search tool. **TB** and **KB** represent the browsing within a tool, such as changing the page of a keyword index, or expanding or contracting a branch of a topic tree. **TR**, **KR** and **SR** represent the final stage of a query, when the content results for that particular method is shown. To insure that all trained models appropriately handle session termination, each initial model used in training has the logout state transition to itself with a probability of one. By guaranteeing that each model ends with a self-transitioning terminal state, one may calculate the expected length of sessions by

determining the expected number of transitions before arriving at the final state. To insure all data handles the terminal state correctly, for every sequence of state transitions generated by of a single user, the **LO** (logout) state is attached twice to the end. To prevent ambiguity about reaching the terminal state, in all initial hidden Markov models used, the logout state can only emit one symbol (**LO**) and can never leave that state.

6.4.2(b) Advanced Vocabulary

The basic vocabulary of two letter codes, while demonstrably useful in distinguishing user types, lacked the flexibility and granularity necessary for many modeling operations. PIVoT/PT offers several tools to access content: search, topic list, keyword list, a textbook index, and Personal Tutor suggestion. Once content is reached, the basic approach does not provide distinct symbols based on how a user reached the content. In order to capitalize on the knowledge that different students prefer different content types, the vocabulary of observation symbols must be designed to distinguish the media format requested (video, text, multiple choice question, etc.), as well as the access method.

To make symbol assignments more powerful and the vocabulary richer, the length of symbols were changed from exactly two characters in the basic vocabulary to between two and five characters in the advanced. Similar operations have similar lengths and begin or end with a common character. For example, all pages not presenting content or search queries are of the form `!`**`XX`**, with **`XX`** representing one of several page codes. In addition to choosing length and initial characters carefully, the positioning of the symbols was chosen carefully. For example, all presentations of content were of the form **`M`**`PR`**`AX`**, where **`M`** is one of the five major media types (`V`, `B`, `S`, `F`, or `Q`) and **`AX`** is one of several two-letter access methods (i.e., `QS` represents a query made from the search page). By purposefully choosing the length and positioning of the characters in the advanced vocabulary, it is possible to wildcard similar symbols easily.

Since content type and access tool are mostly independent, the number of presentation observations is proportional to the number of access tools and content types. This vastly increases the vocabulary from the basic approach. Such a large vocabulary is difficult to work with directly. To solve this, a smaller, model-specific vocabulary is

made by aggregating together symbols that can be treated the same. For example, there exist several symbols for URLs unrelated to content search or presentation. A model may wish to consider all such observations as one. Another model might wish to distinguish one of these observations from the rest, such as the home page. A mapping system was therefore needed to selectively shrink the vocabulary on a per-model basis.

6.4.2(c) Wildcarding System

In Markov and hidden Markov models, the observations are represented by positive integers. A many-to-one mapping between the observation symbols and integers was used for the basic vocabulary. The advanced vocabulary made such a system tedious and hard to interpret, since the set of mappings grew to be rather large. To simplify the mapping of symbols to integers, a wildcarding system was created.

The wildcarding system is similar to the system used in file systems such as UNIX or Windows. The character "`?`" is used to represent any possible single character, where the character "`*`" is used to represent any number of characters, greater than or equal to zero. The two characters can be combined to form a third wildcard, "`?*`" which means any number of characters of length one or higher. When mapping observation codes to integers, one may use either explicit or wildcard symbols. There is the possibility, however, for conflict between explicit and wildcard representations. The resolution is discussed below in section 6.5. It is also possible to remove certain symbols from appearing in the integer sequence: mapping a symbol or wildcard expression to -1 removes the particular request, shortening the integer sequence.

6.5 Implementation

ALBM was designed to create learner models for use both online by the Personal Tutor during user sessions as well as offline for formative analysis. To facilitate server integration and insure ease of use with the rest of pivot, all of ALBM was implemented in Java. ALBM has three main components: the *HMM* class, the *UserSession* class, and the *UserModel* classes.

6.5.1 HMM Classes

The *HMM* class implements all major hidden Markov Model algorithms. The class itself represents a given model, leveraging the sparse and dense matrix and vector classes described in section 4.4.2 above. Developers can construct HMM objects through constructors that take either λ or the model components π, A, and B. For Markov modeling, an identity matrix may be passed on for B, the observation matrix.

In order to solve the three key questions of hidden Markov modeling, inner classes for the Baum-Welch, Q*, and Alpha-Beta calculations were created. These inner classes encapsulate the entire structure of the solution, and all support serialization so that results may be stored in the persistent DB store.

6.5.2 User Session Classes

PIVoT/PT is designed to record all necessary information about a user's session automatically, using the *UserSession* class. This class represents all information about a session, in either the past or currently active. Retrieval methods allow developers to retrieve collections of past sessions that match particular criteria: an individual user, a list of users, or a *TutorletSection* (see section 5.3 above). In addition to past sessions, the Tutorlet API has access to the evolving session of the current user. This allows tutors to train on sessions as they occur.

Each UserSession instance can be queried for key statistics and properties, including its unique identifier within the database, the user, and the session's start, end and elapsed time. Each session also has a list of *UserSession.Request* instances called its trajectory. Each instance of this class, shortened as USR, represents the information pertinent to each server request. An additional class, UserSession.QuizResult represents information on any multiple choice question answered in PIVoT, referring to the particular requests that presented the question, the answer, as well as the score on the quiz and the number of attempts the user made to answer the question.

The UserSession and USR classes contain methods for converting trajectories to observation sequences. Each USR instance can access its two-to-five letter advanced observation code. Additionally, the USR class provides a method for translating an observation code to its appropriate integer given a mapping of explicit and wildcard

symbols (see section 6.4.2 above). Additionally, the UserSession class has methods for converting whole sessions to a trajectory of observation symbols, and with a given map, to sequences of integers.

As discussed above, wildcarding often results in conflicts caused by several mapping keys applying to the same symbol. Conflicts are resolved by the USR class using a hierarchy of wildcard situations that prioritize more specific wildcards over ones that are more generic. The USR class automatically searches the map in this priority order, preventing unintended mappings while preserving the ease and simplicity of the wildcard system.

6.5.3 User Model Classes

To simplify the storage of particular user models, a *UserModel* interface was created. This class abstracts the methods common to all ALBM models, including access to mappings, sequences, and initial HMM guesses. The *UserModel* interface is declared serializable, thus insuring all developers who create user models must make them storable in a file or database.

One class that implements the *UserModel* interface is the *LtoRMarkovModel* class. This class represents the common "left-to-right" Markov model of user behavior. Such models start from an initial state and eventually move to a single termination state from which they never escape. This is appropriate for user modeling since the termination state well-represents logging out. The expected length of a session can also easily be computed with a left-to-right model. This Markov model is also transparent (non-hidden), simplifying analysis. Other implementations are left for future investigators.

6.6 Sample Applications

This section offers a look at sample applications of Adaptive Learner Behavior Modeling. One powerful application of ALBM is in user-preference modeling. The simple yet useful Media Preference Model is discussed, demonstrating how such models can easily be constructed and deployed into a Personal Tutor application. Another powerful application of ALBM is user classification. A simple experiment in user

classification is discussed, which attempts to classify students from different colleges according to stochastic models trained on student trajectory data from each of the colleges. This preliminary experiment lays the groundwork for future work in learning-style preference user classification.

6.6.1 Media Preference Model

Adaptive learner models are easily deployed into a server environment. Observlets can train on existing sessions or past sessions, and interpret these models when making decisions. Since training such models requires time, an Observlet must create a separate thread to perform these computations. Once trained, models can also be serialized so they will be available immediately at the beginning of a new session, without tedious retraining. As newer data is available, models can be retrained in the background and made available to other agents.

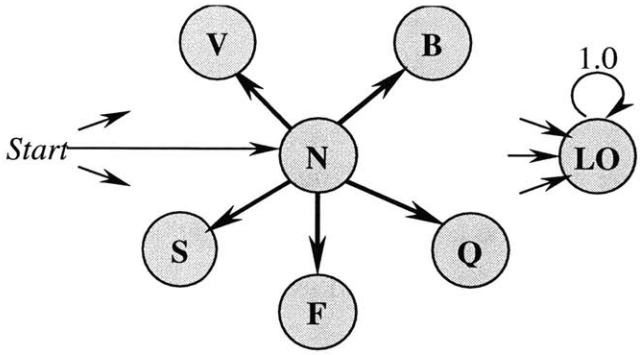

Figure 15: Media Preference Model

A simple example model that was deployed is the Media Preference Model. This model attempts to discover the distribution of media types used over the history of a given user. The model consists of only seven states (see above). Five states (**V**, **B**, **F**, **S**, and **Q**) are assigned to the five types of media that can be presented to the user. The wildcard mapping system simplifies identifying all such pages, since each presentation page is of the form `MP*:*`, where `M` is the letter corresponding to the appropriate media type. The null String, corresponding to all unmapped symbols, is assigned to state six, shown in the diagram as **N**. The **N** state would correspond to all pages not used in presenting content, including starting points such as the home page, search page, result listings, and other "media-neutral" pages. The `!LO:*` code is mapped to the seventh

state, represented in the diagram as **LO.** This terminal state is reached when the user logs out or times out, transitioning with probability one to itself.

While all states have non-zero transition probabilities between them, for clarity, not all transitions are shown in the diagram. Instead, of interest in the media preference problem are the transition probabilities from the **N** state to the five media type states. The probability a user is interested in a particular media type is equal to the conditional probability a user would transition to one of the five media type states, given that the user is in the neutral state. Since this model has very simple goals, the Markov assumption is not a hindrance to realistically modeling the problem. The past states of the user are unnecessary for measuring the probability of transitioning from a neutral state to a media state.

More sophisticated models may wish to more deeply understand the path which users take to a particular media type. This can be done by either aggregating together fewer states via the wildcarding system, or creating an n-step Markov model. An n-step Markov model may offer the greatest flexibility, but doing so increases the number of states in a model exponentially with regard to the finite number of steps remembered. For example, a more sophisticated media preference model may use an n-step approach that would highlight recent transitions learners make from one media type to the next. Such a model may reveal more detail about the learner at the expense of complexity and an exponentially increased state space size.

This model is implemented through the *MediaPrefModel* class, which contains structures that contain the mappings and appropriate initial guesses. Since this model is a left-to-right Markov model, it inherits much of its functionality from the abstract *LtoRMarkovModel* class. Beyond those methods inherited from its superclasses, additional methods were created for interpreting the model as well. For example, a `getMediaPrefDist()` method allows the user to retrieve a probability distribution of media preferences as a five-element vector.

The MediaPrefModel Observlet is used by the *Suggest* agent, described above in section 5.7.4. When a user logs out, the media preference agent creates a thread to recalculate the model, using data from the newly completed session. When the user logs in again, the model is immediately available for use in determining the most appropriate

media types to show to each user. This enhancement to the Suggest system is both probabilistic and dynamic. Its probabilistic nature prevents the agent from always suggesting the same type, since all types have a non-zero probability of being suggested. This need not assume the user has viewed each media type at least once, since the initial training model assumes that the user is equally likely to desire all media types. This model is dynamic in that as the user shifts his preference of media types, the suggestion engine will adjust its suggestions accordingly. This being so, the media preference model achieves the goal of personalization using only simple, Markovian means.

6.6.2 User Classification

As discussed above, one of the primary motivations for the Adaptive Learner Behavior Modeling system was user classification. Several *classes* of users would be identified, perhaps corresponding to each of the different responses to a common question. Training data would be gathered from members of each of the known classes, and used to build stochastic models that best represent the usage data. Stochastic decision processes could then be used on unclassified users to identify the most likely class to which the user belongs, based on sample usage data from one or more sessions of that particular user.

6.6.2(a) Methodology

One easy way to classify users is by their responses to individual multiple-choice questions from a learning style survey, perhaps administered online using a dedicated survey agent. Once such responses have been gathered, and sufficient training data is available, ITS modules can be made to take actions based on stochastic decisions made on unclassified users and trained models. In order to evaluate whether or not such decisions are accurate, unclassified users must also be administered the identical survey. By comparing the agent classification decisions to the unclassified user's survey responses, one can evaluate the efficacy of the stochastic model in learning the pedagogical preferences.

The user classification experiment described above, unfortunately, requires significant training data, as well as significant testing data. Due to data-collection

difficulties described below in section 7.1, it was impossible to gather the needed user participation required to perform the above experiment. Despite this, a similar preliminary experiment was conducted using pre-existing data and known classification categories.

The learner modeling system was also used with the basic vocabulary, shown in Table 1 above in section 6.4.2(a), to aggregate the behavior of users from a given college's introductory mechanics course into one model per college. PIVoT was deployed in the fall of 2000 at three universities: the Massachusetts Institute of Technology (MIT), the Rensselaer Polytechnic Institute (RPI), and Wellesley College. These models were used to stochastically decide which university that a student, randomly chosen from the training data, attended, given only a single trajectory of that student.

Six models were trained for this experiment, consisting of a hidden and non-hidden Markov model for each of the three schools. The initial non-hidden Markov model assumes the user is equally likely to start from, or transfer to, any symbol in the vocabulary. Being a non-Hidden model, the initial observation to state matrix **B** is the identity matrix: it mathematically cannot change during training. The hidden Markov model assumes a number close to one on the diagonal of the observation to state matrix (***B***), and small but positive uniform values for the rest. This approach allows multiple states to correspond to multiple symbols, allowing a potentially more abstract model to evolve. Each of these six models was trained on usage data for PIVoT from the appropriate college. Both approaches are analyzed below.

6.6.2(b) Analysis

This section begins with a qualitative analysis of the Markov (non-hidden) trained models of the MIT and Wellesley students. The RPI students were also similarly analyzed, but since there is no anecdotal data to corroborate usage, only the MIT and Wellesley models are qualitatively analyzed. The RPI student model was used, however, in a three-way quantitative comparison, described later in this section.

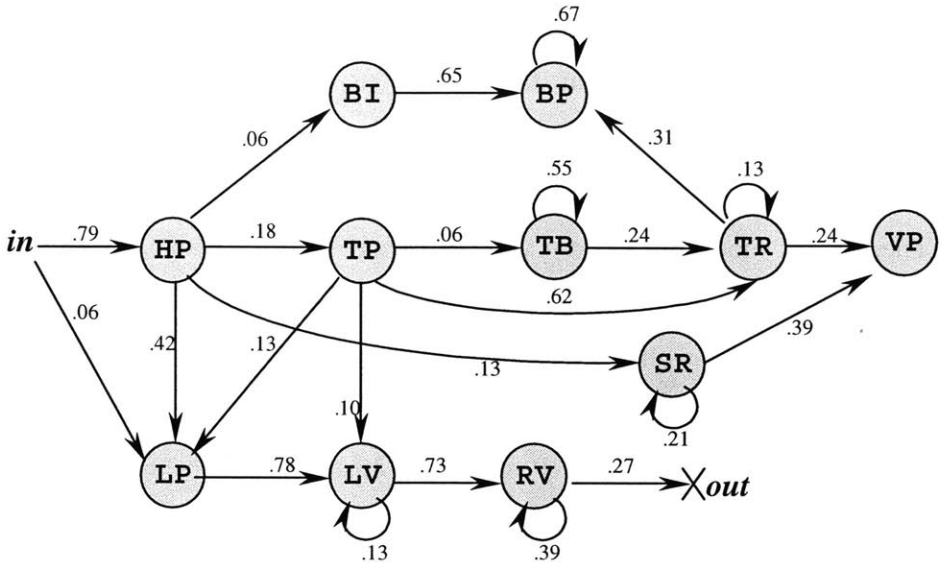

Figure 16: Simplified MIT Markov Model

Though the Wellesley model and MIT model have similarities, attributable to the nature of the PIVoT system itself, there are sharp differences in how the system was used between the two campuses. The Wellesley professor sent emails to students containing links to specific multiple-choice practice problems that she used for grading her students. As students used these email links, it became possible to start PIVoT directly from within a quiz, bypassing the PIVoT home page.

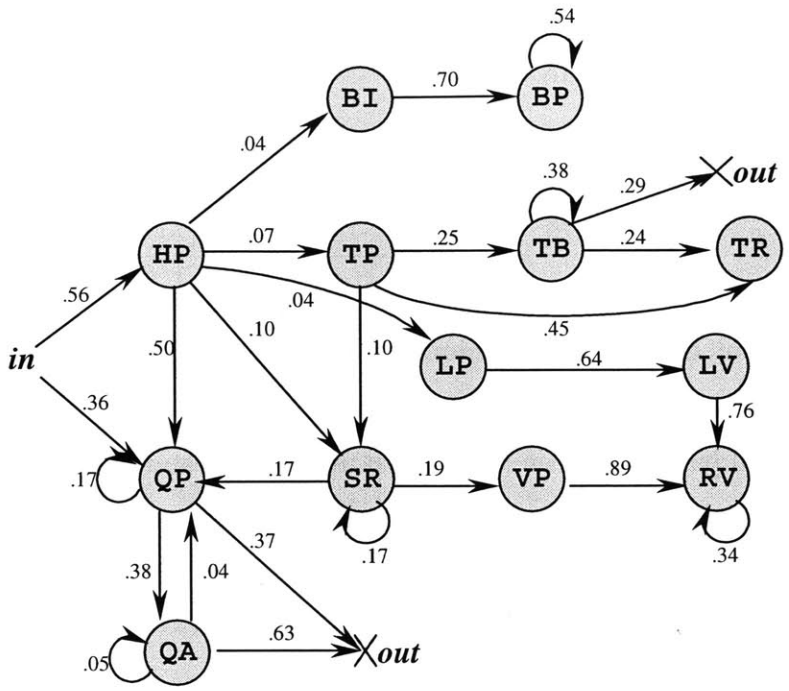

Figure 17: Simplified Wellesley Markov Model

Due to the frequency with which Wellesley students use the "email quiz entry" into PIVoT, one notices a large minority (36%) of students begin their session directly from quizzes, as opposed to the normal home page that typical PIVoT users see. The home page is the starting point for 79% of MIT students (see Figure 16 above), yet for only 56% of Wellesley students (see Figure 17 above). Additionally, 63% of Wellesley users end their session after taking the quiz. Thus, stochastic models can be used to easily distinguish MIT from Wellesley students. Table 2 below shows distinguishing accuracies of approximately 87%.

Table 2: MIT and Wellesley Markov Accuracy

Markov Models	Chose MIT	Chose Wellesley	Accuracy
MIT Data	**886**	144	86.0%
Wellesley Data	174	**1478**	89.5%

Another notable observation from the Wellesley non-Hidden model is that Wellesley students often found the Topic Tree tool confusing. Almost half (45%) the time the typical Wellesley users never expand the topic tree before gathering results (see Figure 17 above). Since very few results are at the top-level topics, this indicates a possible design flaw in the graphical user interface. MIT students seem to be even more

confused by the Topic Tool, with only 6% of users clicking deeper into the topic tree after coming to it (see Figure 16 above).

Table 3: MIT and Wellesley Hidden Markov Accuracy

Hidden Markov Models	Chose MIT	Chose Wellesley	Accuracy
MIT Data	**873**	157	84.8%
Wellesley Data	145	**1507**	91.2%

Another sign of the frustration unique to Wellesley students is that the typical Wellesley student exits PIVoT 29% of the time upon browsing in the Topic Tool, without ever coming to a result page. One also notices that the search feature is used a lot by both MIT and Wellesley students, while the Keyword Index is rarely used. Anecdotal information from the spring 2002 focus group (see section 7.2 below) confirms the quantitatively derived observations made here, made nearly two years prior to the focus group study described in Chapter 7 below. Formative assessment data such as this can be invaluable to user interface designers.

Table 4: MIT, Wellesley, and RPI Markov Accuracy

	Chose MIT	Chose Wellesley	Chose RPI	Accuracy
MIT Data	**677**	57	296	65.7%
Wellesley Data	116	**1292**	244	78.2%
RPI Data	177	24	**707**	77.9%

The hidden Markov model seemed to be more accurate at recognizing Wellesley students, but less accurate regarding MIT students (see Table 3 above). Certain model states tend to aggregate similarly purposed user states, such as the various presentation states (in particular FAQ and Book presentation observations.) The clarity of state assignments found in the non-hidden Markov model however, proved more useful in qualitatively understanding the differences of behavior of the typical PIVoT user from different campuses.

Table 5: MIT, Wellesley, and RPI Hidden Markov Accuracy

	Chose MIT	Chose Wellesley	Chose RPI	Accuracy
MIT Data	**670**	58	302	65.0%
Wellesley Data	103	**1307**	242	79.1%
RPI Data	179	25	**704**	77.5%

In addition to classifying MIT and Wellesley by both MM and HMM strategies, classification was attempted for the three-class problem. The same classification experiment was run using all three student models: MIT, Wellesley, and RPI. Not surprisingly, the accuracy is lowered, but is still accurate from 65% to 79% of the time (see Table 6 below). HMM models are similarly accurate, as expected (see Table 7 below).

Table 6: MIT, Wellesley, and RPI Markov Accuracy

	Chose MIT	Chose Wellesley	Chose RPI	Accuracy
MIT Data	**677**	57	296	65.7%
Wellesley Data	116	**1292**	244	78.2%
RPI Data	177	24	**707**	77.9%

6.6.2(c) Conclusions

Note that with this project the highest accuracy rate achieved in classifying students in this problem is approximately 90%. Higher rates of accuracy are hard to achieve due to the limited nature of the data sets given. For example, many sessions are short. Some sessions are simply ended immediately after beginning, with the user going straight to the exit. These sequences are common for all three classes, and thus are indistinguishable. However, one could imagine trying to identify a user by multiple sessions. By using multiple sessions, the decisions would be made using more data, and likely would increase the accuracy of classification. Nevertheless, it should be noted that the accuracy rates achieved here ought to be more than sufficient for classifying users by pedagogical preference, where the penalty for error is trivial: when a sub-optimal teaching strategy is chosen, the consequences are minor.

Table 7: MIT, Wellesley, and RPI Hidden Markov Accuracy

	Chose MIT	Chose Wellesley	Chose RPI	Accuracy
MIT Data	**670**	58	302	65.0%
Wellesley Data	103	**1307**	242	79.1%
RPI Data	179	25	**704**	77.5%

6.7 Summary

The Adaptive Learner Behavior Modeling system described in this chapter provides technology for creating powerful stochastic models that operate on simple, observable surface-level usage patterns. Through an advanced vocabulary and wildcarding system, developers can easily aggregate different user states to target particular user behaviors.

ALBM can also be used to formatively assess the effectiveness of Web-based interfaces. ALBM models may reveal qualitative details of user behavior through simple, quantitative analyses; by revealing site feature usage preferences, one may confirm or dispute statistically what is told anecdotally. One of the more powerful uses of ALBM is to focus development resources to the most frequently used aspects of the user interface. Alternatively, if certain interface features are being ignored, one might conduct a focus group to discover the reasons behind users' avoidance of these features. ALBM could help center and shape the discussion since it would already be known which features were the most underutilized.

7 Preliminary Data

This chapter offers a glimpse of how PIVoT and the Personal Tutor have been used in an academic setting. This chapter begins with a history of early data collection attempts, revealing the evolution of PIVoT and the Personal Tutor, leading up to the Personal Tutor focus group. In addition to a discussion of the composition, protocols and results of the focus group, a survey of application modules gives insight both into the nature of the tested system, as well as possible future applications of the Tutorlet API. Since difficulties in achieving widespread usage of PIVoT and the Personal Tutor prevented obtaining statistically significant results, the preliminary data presented here centers on the individual experiences of a small focus group. This chapter also discusses preliminary results of applying learner behavior modeling to user classification for a wide deployment of PIVoT before the Personal Tutor was fully deployed.

7.1 Early Data Collection Attempts

As discussed earlier, PIVoT and the Personal Tutor were designed to allow both to be developed separately. This two-track method allowed both projects to progress and evolve at different rates. Despite the freedom to extend and enhance both in parallel, the utility of the Personal Tutor depended on the power of PIVoT and the modules that simplify the use of the ontology. This being so, early development efforts were focused on the PIVoT architecture and presentation tools, at the expense of PT. This emphasis resulted in slowing the integration of new Personal Tutor features into PIVoT.

When PIVoT was deployed in fall of 1999, the Personal Tutor was still in the most nascent stage of development. Additionally, search methods for PIVoT content were a very small subset of what is available at the time of this writing. In order to

access content, PIVoT users had to identify a single keyword and browse through an online list (the "Keyword Tool") to view related media. Text-based searches and topic browsing were still in development. Nevertheless, Professor Lewin integrated PIVoT into the curriculum for *Physics I* (better known at MIT by its course number 8.01) that year: Homework assignments referred to content in PIVoT, often by referring to the keyword and name of the content. While this scheme was somewhat awkward, it was effective in directing students to relevant information.

By the fall of 2000, PIVoT had undergone many improvements. In addition to browsing via the Keyword Tool, the new Topic Tool provided an expanding and collapsing topic tree that allowed students to search by topic. In addition, text-based search boxes were added to each page, using the new query capability added to the PIVoT domain-independent ontology (DIO). Professor Edward Farhi, now teaching *Physics I* in fall of 2000, did not integrate PIVoT into the curriculum. While usage was voluntary in both 1999 and 2000, homework assignments no longer referred to PIVoT content, resulting in less incentive to use PIVoT [Lipson 2001, 5]. Despite this, ninety percent of students who did use PIVoT found it to be an excellent supplementary learning aid [2001, 18].

By the time the fall 2000 version of PIVoT was deployed, the Personal Tutor supported only a single-agent model. As discussed in Chapter 5, this approach complicated application development. Lacking events and a messaging model, it became difficult to integrate multiple features into a single system. Despite this, an early experimental version of the context and suggestion mechanism was deployed. With much of the Personal Tutor in development, no formal data collection plan was undertaken.

By fall of 2001, development on PIVoT had been completed, and focus gradually shifted toward creating a robust Personal Tutor. Most of the Personal Tutor was ready for fall of 2001, but a lack of integration of PIVoT into the course, or use of the Personal Tutor by the faculty, hindered any data collection efforts. Professor Farhi taught *Physics I* again in the fall semester of 2001, using the traditional large lecture/small recitation format. PIVoT was briefly advertised to students at the beginning of the term, but was not utilized by the professor or the teaching staff. Several introductory sessions were

held to recruit members of the recitation staff to use PIVoT and the Personal Tutor in their sections, yet less than ten percent of the staff took the first, short, step of creating a PIVoT account. As the literature in education technology confirms, adoption rates are impaired without strong faculty support, endorsement, and integration into the curriculum [Bates 2000, 121]. Without usage in the classroom, few of the students made the effort to explore and use PIVoT on their own.

Other attempts were made to collect usage data in fall of 2001. MIT offers an alternative freshman education program known as Concourse. One teaching assistant was selected for evaluating PIVoT, the Personal Tutor, and the *Tour* system for communicating between faculty and students. While several Concourse students signed up for PIVoT accounts, the anticipated commitment from the instructor never materialized, and data collection became impossible.

Another attempt to work closely with an instructor in Wellesley College's equivalent physics course was hindered by video performance issues. PIVoT has been used outside of MIT in the years the project has been deployed at both Wellesley College and the Rensselaer Polytechnic Institute (RPI). Whereas RPI had a mirror version of the PIVoT Web and video server, Wellesley users accessed the same servers as MIT users. Wellesley College, being off the MIT campus network, encountered major network bottlenecks when multiple students attempted to view video content simultaneously. Therefore, while some initial attempts by the Wellesley instructor to create Tours were successful, the poor performance resulted in the instructor deciding to curtail PIVoT usage in the classroom.

7.2 Personal Tutor Focus Group

After unsuccessful attempts to garner voluntary faculty use of the Personal Tutor in the classroom, it was decided that a smaller-scale approach to data-collection would be most practical. Discussions, interviews, and written surveys with a select group of PIVoT/PT users could gather valuable qualitative data, which emphasize individual experiences. Since any conclusions would be statistically insignificant, this section only offers qualitative analysis.

It was decided to form a small focus group of students taking the spring 2002 version of *Physics I*. Most students take this course in the fall term since it is both a part of the core curriculum at MIT and a prerequisite for many courses required in several majors. Thus, the overwhelming majority of students who take the spring version of 8.01 have failed the subject in a previous term, typically the preceding fall.

Taking into account the unique makeup of the spring version of the course, the physics department typically uses a different format and pace. For example, since most students have already attended 8.01 in a previous term, traditional thrice-weekly lectures are not offered during the spring term. Instead, lecture is held once each Friday, and used primarily for weekly testing. Students instead attend recitation sections of roughly 20 students each, every Monday, Tuesday, and Wednesday. Students performing poorly are required to attend tutorial on Thursday. Assignments are due three times a week, each substantially shorter than problem sets assigned weekly in a traditional version of 8.01. The uniform distribution of assignments, combined with weekly testing, attempt to prevent students from working on assignments in all-night, last minute sessions and from "cramming" before exams every few weeks.

Professor David Pritchard was the head lecturer for *Physics I* in the spring of 2002, when the focus group was formed. Professor Pritchard is also the coordinator of the CyberTutor research project at MIT. CyberTutor is a Socratic, domain-dependent Web-based intelligent tutoring system. CyberTutor offers procedural help to students; that is, it helps students work a problem through to a correct solution, providing hints or simpler sub-problems along the way. CyberTutor includes several free response answer formats, in contrast to the multiple choice questions used in PIVoT. Students may enter symbolic expressions, fill-in-the-blank, mouse-drawn vectors, and mouse-drawn curves. CyberTutor was first deployed in the experiment-enriched alternate version of *Physics I* each fall, 8.01X. Professor Pritchard, as head lecturer and coordinator, incorporated CyberTutor into the curriculum of spring 8.01. Two of the three weekly homework assignments are electronically submitted using CyberTutor; the third is a traditional pencil-and-paper physics problem set.

7.2.1 Composition and Format

The Personal Tutor Focus Group consisted of five students taking the spring 2002 version of 8.01, taught by Professor David Pritchard. All five were in their second semester at MIT, having failed 8.01 (or one of the alternative versions) in their first semester. Two students, AE and BL, are female. The remaining three, SC, ML, and DM are male.

All students were compensated $150 for their participation in the focus group. Participation required using PIVoT and the Personal Tutor for a minimum of one hour each week and attending three 30-90 minute meetings: an introductory meeting in the first two weeks of the semester, a survey and discussion session in the middle of the term, and individual usage interviews before the final exam.

7.2.2 Registration and Introduction

Before attending the introductory session, all students were required to sign up for an account using their own personal computer or campus workstation. Upon registration, students electronically acknowledged their participation in the group, and took a standardized mechanics aptitude test, the Force Concept Inventory (FCI).[13]

Soon after the beginning of the term, all five students attended the introductory session. Students were introduced to the focus group, informed of their responsibilities, and participated in an informal discussion of their experiences with 8.01 in both the fall and spring. After this discussion, all students convened in an electronic classroom containing several UNIX workstations and a computer projection system at the front of the room. Each student connected to the PIVoT site using each of their accounts, while a tour of the features of PIVoT and the Personal Tutor was given at the front of the room.

The tour of PIVoT and the Personal Tutor showed students all different methods of accessing information, as well as the different media types available. Students were encouraged to use the search engine, the keyword browser, and test using all content

[13] The FCI exam, created by David Hestenes and G. Swackhammer [1992] measures improvements in conceptual understanding of qualitative physics concepts. However, due to the small sample size of the focus group and absence of a control group, the FCI data was not used.

types, including videos, simulations, frequently asked questions, multiple choice questions, and the textbook. In addition, all features of the Personal Tutor were addressed, especially the "Auto Suggest" contextual help feature and the "Test Your Knowledge" sample quiz generator. Where possible, all installation and technical issues that students have experienced or might experience were addressed. The session successfully insured that all students were both familiar with PIVoT and the Personal Tutor, as well as capable of accessing the educational resources, in order to provide meaningful qualitative data.

7.2.3 Group Discussion and Survey

The second meeting of the focus group was held midway through the term. This was the last time the group met as a whole. The 90 minute session was divided into two parts, equal in length: In the first part, students filled out a short survey at identifying learning style preferences and opinions of PIVoT and the Personal Tutor. In the second, students gathered in a conference room for a discussion about their responses and their opinions on PIVoT, the Personal Tutor, and how they used both with 8.01 to date.

The survey used both short multiple choice and numerical ranking questions, as well as open-ended response questions. Students were asked to rank their most useful media types (i.e., videos, simulations, etc.) and to rank their most useful "starting points" (i.e., the search engine, the topic tree, the online text's table of contents). Additionally, students were asked to assess how strongly they agree or disagree with several statements about their learning style and the obtrusiveness of the Personal Tutor (i.e., "I learn best by reading," "I find the Personal Tutor obtrusive," etc.). Finally, the survey offered several open-ended questions on the ease of use of PIVoT, its best, worst, and missing features, how it compares to CyberTutor, and if the student would like to see a system such as PIVoT in their other classes.

While the interviews at the end of the term highlighted individual experiences, the group discussion aimed to discover trends in usage of PIVoT and the Personal Tutor. The discussion suggested that media-type preferences and reasons for use vary most from user to user. These differences were explored individually with each student in the usage interviews. The most common traits involve the time at which the Personal Tutor is used,

the unanimous preference for the search engine as starting point, and the short, purposeful, self-directed nature of PIVoT sessions.

7.2.4 Usage Interviews

At the end of the semester, less than a week before the final, all five focus group participants were given exit interviews. All participants were interviewed individually, in sessions that lasted approximately 25 minutes each. The first ten and final five minutes of each interview consisted of open-ended discussions using a set of scripted questions. The remaining time was spent observing a typical session with PIVoT and the Personal Tutor for that particular student.

In the interview, various questions addressed usage patterns, learning styles, reliability, and comparisons to CyberTutor. Of particular interest was the affect PIVoT has regarding the need for, and use of, face-to-face time with the 8.01 staff. Usage interviews emphasized the individual details of how PIVoT was useful, both in working on assignments, as well as in reviewing for weekly exams. After observing how the individual uses PIVoT to suit his or her needs, the student was then directed to various features of PIVoT and the Personal Tutor for comments on frequency of use and usability.

7.2.5 Results and Analysis

The spring focus group gathered useful information about how a domain-independent supplemental Web resource can aid instruction in the physics domain. This section discusses major usage trends, highlights from individual experiences, and some limited conclusions. All results, however, must be qualified by the limited nature of this preliminary data-collection procedure.

7.2.5(a) Qualifications

Before analyzing the results of the focus group session, it is important to mention several caveats. Firstly, the size of the group prevents any definitive conclusions about usage trends. Secondly, while the group is diverse in both gender and learning styles, the small group is homogeneous in the broader categories of online learners or physics students. All students have failed freshman mechanics for the first time, and all are time-

pressed MIT students who are studying physics because it is required, and not necessarily out of interest.

When observing PIVoT and Personal Tutor usage, it is important to frame all observations around an understanding of the group being evaluated. For example, many PIVoT users are individuals who, being unaffiliated with MIT, sign up for guest accounts to explore physics. These distance learners have a strong interest in the curriculum. While most sessions for focus group members (and 8.01 students at large) are short and purposeful, statistics and anecdotal data from the users themselves indicate that distance learners have longer, explorative, sessions. Since the educational objectives and motivations for using PIVoT are different in both classes, it is reasonable to assume that behavior and pedagogical impact would be different in both groups.

Despite the cautions against overgeneralization, valuable information can be obtained here. Individual experiences within the focus group highlight the diversity of learning styles, as well as innovative ways of using a resource beyond its initial design. In addition, certain universal or near-universal responses are discussed as they provide insight into the design, use, and assessment of Web-based educational resources.

7.2.5(b) Trends

While at some point all content was used by all members, the most popular content was the online text. Many students benefited from the fact that the textbook used in PIVoT was different than the one used in class, giving students "another resource" showing "alternate ways of solving the same problem." Although some students preferred the tangibility of a real textbook, most students appreciated the convenience of an electronic text that was available from any location on campus. Additionally, having the textbook online helped "de-clutter" the desks of students.

Of the five focus group participants, two students used videos extensively. SC, who took an alternate version of 8.01 in the fall term, was never exposed to the demonstrations given in a typical 8.01 lecture. He would search each session seeking videos on the topics at hand, watching shorter help clips first, and eventually working toward the longer videos, such as lectures, available on PIVoT. SC also spoke highly of Walter Lewin as a lecturer, and enjoyed having a different explanation of the material

than the ones received in class. BL also used video to compensate for the different format of spring 8.01. She enjoyed the ability to watch the lectures each week to "feel like I'm getting one." The ability to asynchronously access the lectures of exemplar professors is one of the most compelling features of systems like PIVoT.

BL also made extensive use of PIVoT's online simulations. She found them more understandable than the "two-dimensional" drawings of collisions drawn on the board in class by her instructor. Acknowledging that PIVoT's simulations are two-dimensional as well, she corrected herself, adding that the "interactivity" in the simulations made them effective teaching tools. This hints at how electronic searchable simulations may give meaningful assistance to students who benefit from "hands-on" interactions with important concepts of a particular domain.

The multiple-choice questions available on PIVoT were used by several of the students. They appreciated the hints offered when students chose particular wrong answers. Not all students made frequent use of PIVoT's practice problems since these students tended to use PIVoT to explore material, and not necessarily test knowledge.

All students, regardless of media preference, chose to begin their session by typing a query into a search input boxes, and scan through results. PIVoT offers many starting points to access content at MIT, among them a keyword index, a topic tree, a textbook table of contents, and a listing of digitized lectures. Interestingly, these access points are almost never used. Instead, all students looked for content almost exclusively using the search form. Students began each session having a clear understanding of the words relevant to their problem, and the easiest, most direct way to begin looking for content is the search. If students do not find what they are looking for on the first query, all students search again with a new query; they almost never turn to alternative access points.

The use of the search engine is consistent with the "in and out" nature of PIVoT sessions. As a supplementary resource, students begin each session with a clear understanding of what they were looking for. Once logged in, they search for what they need, read the relevant book section or watch the appropriate video, and leave. Said DM, "PIVoT gets me immediate results; five or ten minutes and I'm done." Only AE admitted using PIVoT for longer, browsing sessions, but only if she was "frustrated" trying to find

what she needed to successfully complete her CyberTutor assignment. Nevertheless, in most cases she too would use PIVoT in "short burst" sessions.

One objective of PIVoT is to replace some of the face-to-face time spent with the physics staff, such as at office hours. Interestingly, most of the students, despite realizing their difficulties, never attended office hours, in either the fall or the spring term. While reasons for this differ, common reasons include tight schedules, the stigma associated with seeking extra help, difficulties in getting to and from campus, and most commonly, the unavailability of staff during the times students work or study: late at night. Most students recognized that PIVoT and the Personal Tutor are not substitutes for the personalized help in office hours, but they did feel that PIVoT, as another resource, lessens the need for office hours. DM went further stating, "It pretty much replaces office hours. It'd be convenient if every class had it. Because for office hours, you kind of have to make time for it, and then when you do have time for it, you have to physically come to campus, but with PIVoT, if every course had it, whenever you're checking mail you can get help, and people are always on their computers anyway." It is important to note that when asked if staff were available during the times at which they used PIVoT, the answer was universally, "no." Supplemental resources such as PIVoT and the Personal Tutor provide a level of interactivity typically unavailable when students need it most.

One recurring theme involves the relative reliability and performance of PIVoT and CyberTutor. Other than occasional video playback performance, PIVoT was found to be always available and responsive. This was in sharp contrast to CyberTutor. Students were harshly critical of the poor performance of the CyberTutor Web site, which was frequently down and sluggish in responsiveness when up. Students reported avoiding using the non-graded aspects of CyberTutor, such as the optional practice problems, because of sluggish performance when waiting for hints or answers. This further underscores the need for high performance and reliability for supplementary resources: if a resource is not required, students will only use it if it is easy, accessible, and reliable.

A widely unused starting point is the online text's table of contents. While the textbook is the most frequently used content in PIVoT, all students enter the textbook

through the search feature. This is intuitive when one makes the analogy between the online textbook and the physical equivalent. In a real textbook, students use a table of contents when exploring or browsing broadly for information. Alternatively, when looking for a particular topic, students begin with the index at the back of the book. Since most students come to PIVoT already with a clear understanding of what they are looking for, they begin by using the search engine. The short, purposeful nature of PIVoT usage can be leveraged when designing agent-based tutors; developers must create agents that can deduce current interest quickly from little usage data, and use contextual interest information from recent sessions.

Students appreciated PIVoT's non-adversarial relationship with its users, in sharp contrast to CyberTutor. Students resented being penalized for wrong guesses with CyberTutor, and thus were reluctant to use it more than required. Surprisingly, some students felt that PIVoT actually taught students more than CyberTutor. Said DM, "CyberTutor is trying to test you, and take points, but PIVoT is trying to help you learn. With PIVoT, I get the feeling it's trying to teach me. With CyberTutor, I don't get that feeling." This is particularly interesting considering that PIVoT and the Personal Tutor lack the sophisticated procedural rules found in CyberTutor. Since PIVoT was used only as a resource, and not used in grading, students all indicated they would continue to use it even if they were not required to do so; this sentiment was mostly not echoed for CyberTutor.

All students were asked their opinion on the obtrusiveness of the Personal Tutor. All students were familiar with Microsoft's Office Assistant, and all but BL expressed strong feelings of dislike for the agent, calling it "annoying" or "obnoxious." Students did not offer the same negative feelings for the Personal Tutor, calling it "unobtrusive." Several students felt it was too unobtrusive, and too easily be ignored. Students said that the tutor "blended into" the background, and the location at the top of the page is often associated with "things that never change" or banner ads, and thus could too easily be ignored. Two students, however, independently offered the same possible solution to improve the visibility of the Personal Tutor. Each suggested moving the display area to the left side of the text, treating the agent as a "sidebar." These students felt that area is more likely to receive attention. Future research into alternate locations and their effect

on usage may be warranted. Nevertheless, the Personal Tutor was used to at least some extent by all students without registering the negative feedback associated with more aggressive agent interfaces.

7.2.5(c) Individual Experiences

Each of the students used PIVoT and the Personal Tutor in unique ways, customizing the resource to their own learning styles. This section highlights the most interesting individual experiences involving how PIVoT and the Personal Tutor were used.

All students found the content-suggestion feature of the Personal Tutor useful. SC felt it saved time by preventing users from searching again when looking for more content on the current topic. AE stated that the feature "sometimes anticipated where I was going next, or raised a topic I hadn't thought of but wanted to look at anyway (out of curiosity)." The persistence of tutors across sessions proved useful to some students. In her usage interview BL began her PIVoT session by clicking on the suggestion offered by the Personal Tutor. BL explained she often would begin sessions by accessing content related to where she left off last, allowing her to "jumpstart" her session by reminding her of her previous stopping point. AE noticed how "the tutor suggests the opposite of the type you're looking at. For example, a video if you're looking at the book and vice versa." AE proposed using the media preferences to list multiple suggestions based on the media types users use most. This idea was eventually implemented, albeit after the focus group concluded.

As discussed above, most students did not browse for particular content; rather, they searched directly for particular key terms, and upon finding relevant content, left the site. BL anticipated using the more browsable starting points, such as the lecture listing, when studying for the final exam. To her, browsing was more useful when "covering topics more broadly." This further emphasizes the specific nature of students' interests when working on an assignment or reviewing for the weekly exams.

While all students in the focus group use the search feature as their primary starting point, not all students search for content in the same ways. Simple search forms, consisting only of a text box and a submit button, are placed on several PIVoT pages. An

advanced form, allowing students to eliminate media types from the results and adjust the maximum number of hits, is accessible by the "search" button on the navigation bar. The majority of students used the basic search interface because of its prominence on the PIVoT home page, which is the first page a student sees when logging in. Though all students have certain content types they search for, the hassle of having extraneous content is minimal for some, and not for others. For some, they "can always just scroll past" what they do not want. Others prefer to always go to the advanced search page and customize the media type and hit count to reduce the clutter of each search. ALBM media preference modeling can ultimately be used to tailor listings to emphasize the types desired most by students.

Each student had unique methods of using PIVoT's interface to best suit their learning style. For example, the textbook has a feature whereby users may click on equations or figures and they appear enlarged in a separate window. BL extensively used this feature, not for improved legibility, but to keep the equation in the forefront of her mind while working on other assignments. DM opened separate windows for the textbook and the search feature to facilitate easy cross-referencing of terms found in the book. AE would always have PIVoT open as a side reference whenever working on CyberTutor assignments. As the literature has shown, users often utilize resources in ways far different than those intended or imagined by the designer.

One feature of divergence among students is the use of search query rankings. All PIVoT queries rank hits with a percent value corresponding to the relevance of the hit to the query. BL and AE were almost unaware of these rankings, where DM used these to avoid viewing any content he deemed to be too poorly ranked to be relevant, "anything below 50% I don't even look at, since it is probably not going to be related." While most of the students in the group do not use them for choosing content, relevancy rankings still impact what content students tend to watch. Since students admit not always looking "all the way down" when reviewing search listings, the fact that the rankings dictate the order in which media is listed have a great impact in what content is accessed by students.

SC made extensive use of the *Test Your Knowledge* feature of the Personal Tutor. After searching for and viewing videos and textbook sections, he would use this Personal Tutor feature to construct a quiz of relevant questions based on recent interest. He

praised the timesaving convenience of automatically generating quizzes based on recent usage without having to search for questions using keywords. Said SC of *Test Your Knowledge*, "I like it because its relevant to what I've been searching, rather than many questions irrelevant to the topic I'm looking for."

7.2.5(d) Conclusions

Some of the most interesting results of the focus group come from comparing PIVoT to CyberTutor. While both are online tutors that cover the same material, the two have drastically different approaches, which affect how its users view each. ML, echoing a universal sentiment, describes PIVoT as "friendlier." Students value PIVoT's role as a resource, especially in comparison to CyberTutor. Said ML, "PIVoT is like a roommate I have that knows physics. CyberTutor is like a TA that is going to fail me." BL adds, "CyberTutor assumes you know a lot... If I don't know an equation, I'll go to PIVoT. I probably could just go to the book, but it's easier using PIVoT." DM concurs, "PIVoT helps you learn, but CyberTutor penalizes you."

One observation gathered from the focus group is that in terms of complexity, more is not necessarily better. Students had significant time to compare PIVoT and CyberTutor, intentionally or otherwise. While PIVoT is far simpler in logic than CyberTutor, students obtained great value from PIVoT's reliable, friendly, simple nature. While CyberTutor offered valuable procedural hints essential to solving physics problems, the simpler multiple-choice questions with answer-specific comments proved extremely popular with students, especially since students could take these quizzes without any fear of it affecting their grade. The chance for broad applicability of PIVoT-like systems beyond physics is enhanced since this question format can be implemented cheaply and easily for almost all domains.

Indeed, one of the strongest positive sentiments for PIVoT and the Personal Tutor echo responses from the larger survey administered two years before the focus group [Lipson 2001]. Both then and now, students wished for a system like PIVoT in their other classes. The focus group participants would like to see similar systems in all introductory math classes, chemistry, and other classes. AE notes the use of simulations in understanding X-ray diffraction.

Perhaps the strongest positive comment on the use of systems such as PIVoT came from a question asked of all participants near the end of the usage interviews. All students were asked if they would use PIVoT to study for the final, now that their focus group commitments were over. Each individual answered strongly in the affirmative. Said ML, humorously, "I have a stack of 8.01 sample final exams with no solution, so... yes."

8 Discussion

This chapter offers a review of the major technological and pedagogical contributions of PIVoT and the Personal Tutor, followed by a discussion of areas for future work related to this research. The chapter concludes with an analysis of how this research fits into the body of work on intelligent tutoring systems and the larger domain of education technology.

8.1 Contributions

The primary contribution of this research is a platform for developing domain-independent multi-agent intelligent tutoring systems. This platform is a combination of several technologies, building upon the literature in multi-agent systems, artificial intelligence, cognitive science, education theory, and information retrieval.

8.1.1 Technological Contributions

This research builds upon Salton's vector space model (VSM) [1975] for describing the relationship between documents and queries to devise a system for efficiently evolving a quantitative measure of the recent interest of an online user. This system can in turn be used to dynamically suggest the most relevant content from a large media base.

The domain-independent ontology (DIO) provides a system for recording the most important surface-level metadata for educational media in such a way that it can be easily used with intelligent tutoring systems. A major contribution to ITS design is the application of correlation techniques to strengthen the relationship between content and the terms needed to describe content. The media-attribute topology approach not only

gives new approaches to tutor decision making, but also allows domain-experts and students alike to better understand the interconnections of a particular domain.

The Observlet Tutorlet Event model contributes to the literature on multi-agent systems by offering a layered approach to negotiation and user-agent interaction. By offering a hybrid agency model, one may obtain the advantages of both single- and multi-agent designs. The Personal Tutor offers an innovative approach to user-agent interaction, allowing dialogues to develop both between Web hits and across entire sessions. The persistence model helps complete the multi-agent communication system by allowing both communicative and non-communicative agents to share knowledge about the user.

The multi-agent approach to intelligent tutoring system design used by the Observlet Tutorlet Event model emphasizes reactive agents. Observlets tend to operate on simple rules, and typically react by posting events or storing descriptive information in the persistent store. As discussed earlier, the intelligence and complex behavior of a Personal Tutor will "emerge" from the interactions between and among simple agents, and the user. This approach is consistent with Nouvelle AI, and contributes to the literature on reactive agent design.

The Adaptive Learner Behavior Modeling system provides another application of stochastic modeling. By speeding the process of formative assessment through quantitative means, educational software designers may focus their efforts on the areas in most need of development effort. Stochastic models also allow intelligent tutoring systems to learn the pedagogical preferences of users with minimal human intervention. By quantitatively predicting learner preferences, students can receive automatically personalized assistance with minimal delay, encouraging use of valuable, Web-based supplementary resources.

8.1.2 Pedagogical Contributions

By emphasizing domain-independence at all stages of design, this research can be more easily applied to other domains beyond physics. The broader applicability of this work can serve to reduce the cost of developing educational courseware, since one intelligent tutoring system can be developed for several academic subjects.

By creating an intelligent tutoring system platform first, and an actual ITS second, this research serves as a springboard for work by future educational software researchers. Since philosophies of teaching vary heavily, the level of abstraction maintained throughout this thesis enables multiple pedagogical strategies to competitively exist within the same software platform.

In addition to allowing multiple pedagogical strategies to coexist, the simplicity of the technological platform allows less technically sophisticated developers, perhaps those whose primary background is in education, the opportunity to develop meaningful educational software experiments.

The "Tour System" furthers the opportunity of non-technical educators to use education technology. While untested, this system contributes to the literature on annotation and course customization. When used by students, it can be used as an interactive enhancement to the "bookmarking notepad" approach seen in the literature [Beaufils 2000].

8.2 Future Work

As is true for many research endeavors, PIVoT, the Personal Tutor, and the technology behind both have raised far more research questions than those answered. The research efforts discussed in this document have opened the door for future research in several technological and pedagogical domains. This section addresses some of the avenues for future research.

8.2.1 Stochastic Behavior Modeling

Chapter 6 discussed the application of stochastic modeling to surface-level trends in site usage. Surveys can be used to classify users by pedagogical preferences and learning traits. A future use of the ALBM system is to test which traits are most accurately distinguishable by surface-level trends. Such research could be easily conducted, though it would require large-scale deployment in order to get sufficient training data for the underlying Markov models.

As previously discussed, the accuracy of stochastically derived models may be evaluated statistically using the following experiment. A sufficiently large population of

students could be divided randomly into two similarly sized groups. The larger group, or *training* group, would be given a learning-preference survey, and the group would be classified by their responses to each question. In all, an ALBM model would be trained for every response to every question, using the users who responded accordingly in each case. The smaller group, or test group, would then be administered the same survey. Additionally each student in the test group would be classified via Expectation Maximization (EM) using the models created from the training group. For each question, one would evaluate how each test group user's survey response compares to the response predicted by the trained models. By comparing the relative accuracy of modeling each question, one may evaluate the learning traits most easily deduced by surface-level trends.

The above experiment can serve as a springboard for a multitude of future experiments. Knowing which learning traits can most easily be deduced in a particular domain may or may not translate to other domains. Future researchers may investigate which domains produce the most observable surface-level traits. Within a single domain, one may evaluate how variations in user interface design may sharpen or dull the ability to observe surface-level trends.

As the ALBM system evolved, increasingly sophisticated vocabularies were designed to more easily express interesting models of user behavior. One may investigate more universally applicable ways of efficiently describing the vocabulary of application states for Web applications. Additionally, research into automatic ways of deducing the states may reduce the effort needed to model users effectively, in comparison to manual means.

Finally, the Markov and Hidden Markov models demonstrated and tested in this research were simple in design. It is possible with more sophisticated use of state and observation assignments, more powerful conclusions may be drawn from such models. Alternatively, one may discover that a multitude of small models with a limited number of states can be used to make the best decisions on users' interests, preferences, and intentions.

8.2.2 Domain-Independent Ontologies

In order to effectively assist students regardless of domain, a system for efficiently recording the relationships between educational content was developed. Once this metadata was collected for the physics domain, algorithms were developed to maximize the value of this domain knowledge. Future researchers may investigate new algorithms for automatically manipulating this information.

During the annotation process for the content of a particular domain, relationships between media items are manually recorded. Currently, the recording of these relationships, or *cross-references*, is a tedious, labor-intensive process. In addition to being tiresome, it is also difficult to exhaustively record all cross-references. Often, certain topics are more richly connected than others are. Future research may investigate automatic means of "filling in" references using the correlation of manual references to the co-relevance of the content they relate.

One of the contributions of this research is the use of information retrieval techniques in intelligent tutoring systems to discover the relationships between the keywords of a domain. Future research may investigate effective ways of using this knowledge to better teach students. For example, one may show users common words from the vocabulary distribution of a particular page. Such information may have significant applications to constructivist approaches to education. By exposing students to the relationships between keywords, students may strengthen their own conceptual models, potentially improving comprehension.

8.2.3 Intelligent Tutoring System Design

The Tutorlet API has not only created the opportunity for future research in multi-agent intelligent tutoring software design, but simplified the process of designing such software as well. With the existence of a reusable design for domain-independent multi-agent courseware, researchers may more rapidly deploy multi-agent intelligent tutoring systems building up the work of others using the same system. One can imagine a Darwinian evolution of ITS design through the improvement, selection, and rejection of each developed agent, based on their adaptability to multiple domains, and success in teaching the domains in which they were tested.

Preliminary data-collection attempts and large-scale statistical analysis have shown that text-based searching is the preferred starting point for inquiries to PIVoT. This is not surprising consider both the directness of the approach, and the similarity to Web-wide search engines. Future research might investigate ways to improve and enhance search results using agents designed to detect trends and preferences in user interest. This agent-provided information can add contextual information to refine simple searches, or tailor the media or difficulty level to the individual user's current state or preference.

While the Tutorlet API and PIVoT's metadata system were designed to ensure domain-independence, it is possible that new multi-agent systems will use these technologies to produce domain-dependent tutors. While running contrary to the design objectives of this research, such systems may still reduce cost and design effort over completely-domain dependent approaches. With the Messaging API allowing for easy user-agent dialogues and competition between agents for user attention, one can see domain-independent and domain-dependent agents coexisting side-by-side to maximally assist the students who use them.

As originally designed the StudentCentral work area was complemented by a TeacherCentral system to serve a similar function for instructors. The TeacherCentral system was to allow instructors to access information about his or her section, as well as to create, edit and publish tours. Due to difficulties in garnering voluntary use from physics faculty members (discussed in section 7.1 above), work on information modules for the TeacherCentral was abandoned in favor of further development on student-related modules.

Despite being set aside, the TeacherCentral system offered the promise of rapid formative assessment. As originally designed, an instructor may log on to PIVoT, go to the TeacherCentral agent, and request a one-page statistical analysis of the topics and problems most troubling students in the last week. This instructor may print this report out, head to class, and use it to adjust the day's recitation to emphasize and review common problems. By aggregating data anonymously, students embarrassed by the difficulty, or otherwise reticent, may still better the instructor's knowledge of common

misconceptions. In addition, the rapid feedback may serve to improve the quality of limited class time with out-of-class online data.

8.2.4 ITS and Pedagogy

While the preliminary data indicates that both PIVoT and the Personal Tutor have been received positively, wider studies that attempt to assess the effectiveness of the system are needed. Many questions arise about the pedagogical effectiveness of domain-independent tutors. Besides studying their effectiveness in comparison to domain-dependent tutoring systems, future researchers may investigate which students benefit most from systems such as PIVoT/PT. Intuition suggests that these resources might benefit self-motivated students the most, but one might counter-argue that these systems help the least-motivated students.

Another area for significant further research is the Tour system. While the technology was fully implemented, large-scale deployment, with faculty encouragement and participation, is necessary for it to be properly evaluated. As discussed earlier, the Tour system can be used with both constructivist and instructivist pedagogy. Future research may compare the effectiveness of each approach, both in terms of quantitative academic performance, as well as in more qualitative terms, such as student interest and participation.

PIVoT and the Personal Tutor were designed for use in traditional academic classrooms, as a Web-based, interactive supplementary resource. Nevertheless, the potential of these technologies call for investigation of other deployment settings. Having roots in distance-education technologies, PIVoT/PT may be repurposed for use in continuing education. In addition, PIVoT and the Personal Tutor certainly may have use in non-academic educational settings, such as corporate training.

In large companies, it is often necessary to train a large segment of the firm's employees through videotape-based lectures and paper-based testing. Traditionally such training must be done synchronously, with large groups of employees coming to a common room together to begin training. This may often inconvenience both trainers and employees. Adapted for corporate training, PIVoT may offer asynchronous online training as well as testing. Each employee can watch training videos at his or her

convenience and at a comfortable pace. When adequately prepared, each employee may be tested online, also using the PIVoT system. Further assistance may come from agent-based tutors using the multi-agent technologies proposed in this document.

8.3 Conclusions

PIVoT and the Personal Tutor advance the art and science of intelligent tutor system design in multiple ways. Through innovations in agency models, negotiation, and user-agent communication, the Observlet Tutorlet Event architecture simplifies the task of creating intelligent tutoring systems. Innovations in behavioral modeling techniques expedite and simplify formative assessment, as well as bring stochastic methods to Web-based domain-independent development. The separation of site development from agency allow for faster, more independent development of both.

As stated above, PIVoT and the Personal Tutor provide a starting point for new research in education technology. Future researchers may use the technological architectures and pedagogical approaches described herein to advance the state of art in artificial intelligence, cognitive science, education, and interface design. The lessons learned by these future endeavors may further our understanding of the relationship between education and technology as a whole.

Bibliography

Agent Technology

[Brenner 1998]

Brenner, Walter. "Agents as Tools of the Information Society." In *Intelligent Software Agents*, by Walter Brenner, Rüdiger Zarnekow, and Hartmut Wittig, in cooperation with Claudia Schubert, translated by Anthony S. Rudd, 7-18. Berlin: Springer, 1998.

[Brenner *et al.* 1998]

Brenner, Walter, Rüdiger Zarnekow, and Hartmut Wittig. *Intelligent Software Agents*. In cooperation with Claudia Schubert. Translated by Anthony S. Rudd. Berlin: Springer, 1998.

[Haugeneder and Steiner 1998]

Haugeneder, H., and D. Steiner. "Co-operating Agents: Concepts and Applications." In *Agent Technology: Foundations, Applications, and Markets*, eds. Nicholas R. Jennings and Michael J. Wooldridge, 175-202. Berlin: Springer, 1998.

[Jennings and Wooldridge 1998]

Jennings, Nicholas R., and Michael J. Wooldridge, eds. *Agent Technology: Foundations, Applications, and Markets*. Berlin: Springer, 1998.

[Nwana and Ndumu 1998]

Nwana, H. S., and D. T. Ndumu. "A Brief Introduction to Software Agent Technology." In *Agent Technology: Foundations, Applications, and Markets*, eds. Nicholas R. Jennings and Michael J. Wooldridge, 29-47. Berlin: Springer, 1998.

[Zarnekow 1998a]

Zarnekow, Rüdiger. "Fundamental Concepts of Intelligent Software Agents." In *Intelligent Software Agents*, by Walter Brenner, Rüdiger Zarnekow, and Hartmut Wittig, in cooperation with Claudia Schubert, translated by Anthony S. Rudd, 19-34. Berlin: Springer, 1998.

[Zarnekow and Wittig 1998]

Zarnekow, Rüdiger, and Hartmut Wittig. "Base Modules of Agent Systems." In *Intelligent Software Agents*, by Walter Brenner, Rüdiger Zarnekow, and Hartmut Wittig, in cooperation with Claudia Schubert, translated by Anthony S. Rudd, 153-188. Berlin: Springer, 1998.

Cognitive Science and Pedagogy in ITS Design

[Albacete and VanLehn 2000]

Albacete, Patricia L., and Kurt VanLehn. "The Conceptual Helper: An Intelligent Tutoring System for Teaching Fundamental Physics Concepts." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 564-573. Berlin: Springer, 2000.

[Albalooshi and Alkhalifa 2001]

Albalooshi, Fawzi, and Eshaa M. Alkhalifa. "Multi-Media as a Cognitive Tool: Towards a Multi-Media ITS" *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 231-234.

[Aleven and Koedinger 2000]

Aleven, Vincent, and Kenneth R. Koedinger. "Limitations of Student Control: Do Students Know When they Need Help?" In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 292-303. Berlin: Springer, 2000.

[Baylor 2001]

Baylor, Amy L. "Cognitive Requirements for Agent-Based Learning Environments." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 462-463.

[Boulay 2000]

Boulay, Benedict du. "Can We Learn from ITSs?" In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 9-17. Berlin: Springer, 2000.

[Buchanan 2000]

Buchanan, T. "The Efficacy of a World-Wide Web Mediated Formative Assessment." *Journal of Computer Assisted Learning* 16 (2000): 193-200.

[Hestenes *et al.* 1992]

Hestenes, David, M. Wells, and G. Swackhammer. "Force Concept Inventory." *The Physics Teacher* 30, no. 3 (March 1992): 141-158.

[Hoole *et al.* 2002]

Hoole, Dushyanthi, N. Yogendran, S. Thavachandran, P. Priyatharshan, and S. R. H. Hoole. "A Bank of Chemistry Questions on an On-Line Server." *Journal of Science Education and Technology* 11, no. 1 (March 2002): 9-13.

[William and Black 1996]

William, D., and P. Black. "Meanings and consequences: a basis for distinguishing formative and summative functions of assessment?" *British Educational Research* Journal 22 (1996): 537-548. Quoted in T. Buchanan, "The efficacy of a World-Wide Web mediated formative assessment," *Journal of Computer Assisted Learning* 16 (2000): 193-194.

Cognitive Science Foundations of Instruction

[Anderson 1983]

Anderson, John Robert. *The Architecture of Cognition.* 1983. Reprint, Mahwah, NJ: Lawrence Erlbaum Associates, 1996.

[Ceci and Ruiz 1993]

Ceci, Stephen J., and Ana I. Ruiz. "Inserting Context into our Thinking About Thinking: Implications for a Theory of Everyday Intelligent Behavior." In *Cognitive Science: Foundations of Instruction*, ed. Mitchell Rabinowitz, 173-188. Hillsdale, NJ: Lawrence Erlbaum Associates, 1993.

[Colaric 2000]

Colaric, Susan M. "The Instructional Systems Process." Internet URL: http://www.personal.psu.edu/users/s/m/smc258/KB/index.html (2000).

[Graesser *et al.* 1993]

Graesser, Arthur C., Natalie K. Person, and John Huber. "Question Asking During Tutoring and in the Design of Educational Software." In *Cognitive Science: Foundations of Instruction*, ed. Mitchell Rabinowitz, 149-172. Hillsdale, NJ: Lawrence Erlbaum Associates, 1993.

[Luger 1994]

Luger, George F., *Cognitive Science: The Science of Intelligent Systems.* San Diego, CA: Academic Press, 1994.

[Thomas and Rohwer 1993]

Thomas, John W., and William D. Rohwer, Jr. "Proficient Autonomous Learning: Problems and Prospects." In *Cognitive Science: Foundations of Instruction*, ed. Mitchell Rabinowitz, 1-32. Hillsdale, NJ: Lawrence Erlbaum Associates, 1993.

[Todd and Morris 1995]

Todd, James T., and Edward K. Morris, eds. *Modern Perspectives on B.F. Skinner and Contemporary Behaviorism.* Westport, CT: Greenwood Press, 1995.

[Wadsworth 1996]

Wadsworth, B. J. *Piaget's Theory of Cognitive and Affective Development.* White Plains, NY: Longman, 1996.

Foundations of Artificial Intelligence

[Brooks 1986]

Brooks, R. A. "A Robust Layered Control System for a Mobile Robot." *IEEE Journal of Robotics and Automations* 2(1), 14-23. Cited in H. S. Nwana and D. T. Ndumu, "A Brief Introduction to Software Agent Technology," in *Agent Technology: Foundations, Applications, and Markets*, eds. Nicholas R. Jennings and Michael J. Wooldridge, 29-47. Berlin: Springer, 1998.

[Maes 1990]

Maes, P. "Guest Editorial." *Robotics and Autonomous Systems* 6 (1990):1-2. Quoted in *Artificial Intelligence: A New Synthesis*, by Nils J. Nilsson, 7. San Francisco, CA: Morgan Kaufmann Publishers, 1998.

[Nilsson 1998]

Nilsson, Nils J. *Artificial Intelligence: A New Synthesis.* San Francisco, CA: Morgan Kaufmann Publishers, 1998.

[Winston 1992]

Winston, Patrick Henry. *Artificial Intelligence.* Third edition. Reading, MA: Addison-Wesley, 1992.

Interface Agents and Adaptability

[Brown and Meyer 1961]

Brown, R. G., and F. F. Meyer. "The Fundamental Theorem of Exponential Smoothing" *Operations Research* 9 (1961): 673-685.

[Ciancarini *et al.* 1999]

Ciancarini, P., Robert Tolksdorf, and F. Vitali. "The World Wide Web as a Place for Agents" In *Artificial Intelligence Today: Recent Trends and Developments*, eds. Michael J. Wooldridge and Manuela Veloso, 175-193. Lecture Notes in Computer Science. Lecture Notes in Artificial Intelligence; 1600. Berlin: Springer, 1999.

[Dimitrov and Warren 1999]

Dimitrov, Dimitre, and James Warren. "More Efficient Web Searching by Modeling Users' Long-Term Goals." In *Intelligent Agent Technology: Systems, Methodologies, and Tools*, eds. Jiming Liu and Ning Zhong, 342-346. Singapore: World Scientific, 1999.

[Han and Kim 1999]

Han, Junggee, and Juntae Kim. "A Personalized Web Agent with Implicit Feedback and Hybrid Filtering Strategy." In *Intelligent Agent Technology: Systems, Methodologies, and Tools*, eds. Jiming Liu and Ning Zhong, 286-295. Singapore: World Scientific, 1999.

[Jee-Haeng Lee and Cho 1999]

Jee-Haeng Lee and Sung-Bae Cho. "Information Retrieval Agents Based on Evolutionary Computation to Reflect User Preference." In *Intelligent Agent Technology: Systems, Methodologies, and Tools*, eds. Jiming Liu and Ning Zhong, 347-351. Singapore: World Scientific, 1999.

[Kozierok and Maes 1993]

Kozierok, R., and Maes, P. "A Learning Interface Agent for Scheduling Meetings." In *Proceedings ACM-SIGCHI International Workshop on Intelligent User Interfaces*, Florida, 81-93. Cited in "A Brief Introduction to Software Agent Technology," in *Agent Technology: Foundations, Applications, and Markets*, by H. S. Nwana and D. T. Ndumu, eds. Nicholas R. Jennings and Michael J. Wooldridge, 29-47. Berlin: Springer, 1998.

[Lieberman 1995]

Lieberman, H. "Letzia: An Agent that Assists Web Browsing." In *Proceedings IJCAI 95, AAAI Press.* Cited in "A Brief Introduction to Software Agent Technology," in *Agent Technology: Foundations, Applications, and Markets*, by H. S. Nwana and D. T. Ndumu, eds. Nicholas R. Jennings and Michael J. Wooldridge, 29-47. Berlin: Springer, 1998.

[Maes 1994]

Maes, P. "Agents that Reduce Work and Information Overload." *Communications of the ACM* 37, no. 7 (July 1994): 31-40.

[Maes 1995]

Maes, P. "Intelligent Software." *Scientific American* 273, no. 3 (September 1995) 84-86.

[Pazzani and Billsus 2002]

Pazzani, Michael J., and Daniel Billsus. "Adaptive Web Site Agents." *Autonomous Agents and Multi-Agent Systems* 5 (2002): 205-218.

[Salton *et al.* 1975]

Salton, G., A. Wong, and C. S. Yang. "A Vector Space Model for Automatic Indexing." *Communications of the ACM* 18, no. 11 (November 1975): 613-620.

[Young *et al.* 1999]

Young, Jun, Hae So, Baek Jung, and Tack Park Young. "User Profile Based Personalized Web Agent." In *Intelligent Agent Technology: Systems, Methodologies, and Tools*, eds. Jiming Liu and Ning Zhong, 296-305. Singapore: World Scientific, 1999.

ITS Models, Frameworks, and Paradigms

[Belkada *et al.* 2001]

Belkada, Safia, and Alexandra I. Cristea, and Toshio Okamoto. "Measuring Knowledge Transfer Skills by Using Constrained-Student Modeler Autonomous Agent." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 375-378.

[Chien-Sing Lee and Singh 2001]

Chien-Sing Lee, and Yashwant Prasad Singh. "A Case-based Agent Framework for Adaptive Learning." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 235-238.

[Devedzic 2000]

Devedzic, Vladan. "Applying Patterns to ITS Architectures." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 123-132. Berlin: Springer, 2000.

[Galeev *et al.* 2001]

Galeev, I. K., S. A. Sosnovsky, and V. I. Chepegin. "ITS Design Technology for the Broad Class of Domains." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 431-432.

[Gertner and VanLehn 2000]

Gertner, Abigail S., and Kurt VanLehn. "Andes: A Coached Problem Solving Environment for Physics." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 133-142. Berlin: Springer, 2000.

[Lelouche 2000a]

Lelouche, Ruddy. "Using Agent Technology to Model Tutoring Knowledge in an ITS." *International Workshop on Advanced Learning Technologies, 2000* (Palmerston North, New Zealand: IEEE, 2000): 33-34.

[Lelouche 2000b]

Lelouche, Ruddy. "A Collection of Pedagogical Agents for Intelligent Educational Systems." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 143-152. Berlin: Springer, 2000.

[McCalla *et al.* 2000]

McCalla, Gordon, Julita Vassileva, Jim Greer and Susan Bull. "Active Learner Modelling." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 53-62. Berlin: Springer, 2000.

[Murray and VanLehn 2000]

Murray, R. Charles, and Kurt VanLehn. "*DT Tutor*: A Decision-Theoretic, Dynamic Approach for Optimal Selection of Tutorial Actions." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 153-162. Berlin: Springer, 2000.

[Papaterpos *et al.* 2001]

Papaterpos, Christos M., Nektarios P. Georgantis, and Theodore S. Papatheodorou. "An Ontology for Modeling Ill-Structured Domains in Intelligent Educational Systems." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 41-42.

[Patel and Kinshuk 1996]

Patel, Ashok, and Kinshuk. "Intelligent tutoring tools-a problem solving framework for learning and assessment." *Frontiers in Education Conference, 1996.* Vol. 1 (Salt Lake City, UT, USA: IEEE, 1996): 140-144.

[Peng-Kiat Pek and Kim-Leng Poh 2000]

Peng-Kiat Pek and Kim-Leng Poh. "Framework of a Decision-Theoretic Tutoring System for Learning of Mechanics" *Journal of Science Education and Technology* 9, no. 4 (2000): 343-356.

[Prentzas *et al.* 2001]

Prentzas, Jim, Ioannis Hatzilygeroudis, and C. Koutsojannis. "A Web-Based ITS Controlled by a Hybrid Expert System." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 239-240.

[Rosselle and Grandbastien 2000]

Rosselle, Marilyne, and Monique Grandbastien. "Experimenting Features from Distinct Software Components on a Single Platform." *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 163-172. Berlin: Springer, 2000.

ITS or MAS Infrastructure

[Badjonski *et al.* 1997]

Badjonski, Mihal, Mirjana Ivanović, and Zoran Budimac. "Intelligent Tutoring System as Multiagent System." *IEEE Conference on Intelligent Processing Systems, 1997.* Vol. 1 (Beijing: IEEE, 1997): 871-875.

[Gasser 2001]

Gasser, Les. "MAS Infrastructure: Definitions, Needs, and Prospects." In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, International Workshop on Infrastructure for Scalable Multi-Agent Systems, eds. Tom Wagner and Omer Rana, 1-11. Barcelona, Spain: June 2000. Lecture Notes in Computer Science. Lecture Notes in Artificial Intelligence; 1887. Berlin: Springer, 2001.

[Heift and Nicholson 2000]

Heift, Trude, and Devlan Nicholson. "Theoretical and Practical Considerations for Web-Based Intelligent Language Tutoring Systems." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 354-362. Berlin: Springer, 2000.

[Lind 2001]

Lind, Jürgen. *Iterative Software Engineering for Multiagent Systems: The MASSIVE Method.* Lecture Notes in Computer Science. Lecture Notes in Artificial Intelligence; 1994. Berlin: Springer, 2001.

[Vincent *et al.* 2001]

Vincent, Regis, Bryan Horling, and Victor Lesser. "An Agent Infrastructure to Build and Evaluate Multi-Agent Systems: The Java Agent Framework and Multi-Agent System Simulator." In *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, International Workshop on Infrastructure for Scalable Multi-Agent Systems, eds. Tom Wagner and Omer Rana, 102-127. Barcelona, Spain: June 2000. Lecture Notes in Computer Science. Lecture Notes in Artificial Intelligence; 1887. Berlin: Springer, 2001.

[Zarnekow 1998b]

Zarnekow, Rüdiger. "Development Methods and Tools." In *Intelligent Software Agents*, by Walter Brenner, Rüdiger Zarnekow, and Hartmut Wittig, in cooperation with Claudia Schubert, translated by Anthony S. Rudd, 153-188. Berlin: Springer, 1998.

Reusability and Domain Independence

[ADL 2001]

ADL (Advanced Distributed Learning). "SCORM: Shareable Content Object Reference Model." Version 1.2. Internet URL: http://www.adlnet.org/ADLDOCS/Other/SCORM_1.2_PDF.zip (October 2001).

[AICC 1998]

AICC (Aviation Industry CBT Committee). "Web-based Computer Managed Instruction (CMI)." Version 1.0. Internet URL: http://www.aicc.org/docs/AGRs/agr010v1rtf.zip (September 1998).

[Becta 2001]

British Educational Communications and Technology Agency (Becta). "Metadata in Education." Internet URL: http://www.becta.org.uk/technology/infosheets/pdf/metadata.pdf (2001).

[CoMPIO 2001a]

Consequence Management Program Integration Office (CoMPIO). "History of Intelligent Tutoring Systems." Carnegie Mellon University. Internet URL: http://www.lsal.cmu.edu/lsal/expertise/projects/compio/compio2001finalreport/intell_tutors/history_it.html (2001).

[CoMPIO 2001b]

Consequence Management Program Integration Office (CoMPIO). "Sharable Content Object Reference Model (SCORM) in the Eyes of Instructional Designers: Past and Future." Carnegie Mellon University. Internet URL: http://www.lsal.cmu.edu/lsal/expertise/projects/compio/compio2001finalreport/appendixes/pedagog_scorm.html (2001).

[CoMPIO 2001c]

Consequence Management Program Integration Office (CoMPIO). "Extending CSF to Support Adaptive Learning Experiences." Carnegie Mellon University. Internet URL: http://www.lsal.cmu.edu/lsal/expertise/projects/compio/compio2001final report/appendixes/extendedcsf.html (2001).

[IMS 2001]

IMS Global Learning Consortium. "IMS Learning Resource Meta-Data Information Model: Version 1.2.1 Final Specification." Internet URL: http://www.imsproject.org/metadata/imsmdv1p2p1/imsmd_infov1p2p1.html (2001).

[Les *et al.* 1999]

Les, Jan, Geoff Cumming, and Sue Finch. "Agent Systems for Diversity in Human Learning." In *Artificial Intelligence in Education: Open Learning Environments: New Computational Technologies to Support Learning, Exploration and Collaboration*, Frontiers in Artificial Intelligence and Applications, Vol. 50, eds. Susanne P. Lajoie and Martial Vivet, 13-20. Amsterdam: IOS Press, 1999.

[Masthoff and Van Hoe 1996]

Masthoff, J., and Van Hoe, R. "APPEAL: A Multi-Agent Approach to Interactive Learning Environments," in *Distributed Software Agents and Applications*, Sixth European Workshop on Modelling Autonomous Agents in a Multi-Agent World, eds. 77-89. Cited in "Agent Systems for Diversity in Human Learning," in *Artificial Intelligence in Education: Open Learning Environments: New Computational Technologies to Support Learning, Exploration and Collaboration*, Frontiers in Artificial Intelligence and Applications, Vol. 50, by Jan Les, Geoff Cumming, and Sue Finch, eds. Susanne P. Lajoie and Martial Vivet, 13-20. Amsterdam: IOS Press, 1999.

[Mühlhäuser 2000]

Mühlhäuser, Max. "Engineering Web-Based Multimedia Training: Status and Perspective." *International Symposium on Multimedia Software Engineering, 2000* (Taipei, Taiwan: IEEE, 2000): 3-12.

[Nakabayashi *et al.* 2001]

Nakabayashi, Kiyoshi, Yukihiro Kubota, Hiroshi Yoshida, and Tatsuo Shinohara. "Design and Implementation of WBT System Components and Test Tools for WBT Content Standards." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 213-214.

[Paiva *et al.* 1999]

Paiva, Ana, Isabel Machado, and Alexandre Martins. "Using and Re-using Agents in Multi-Agent Learning Environments." In Artificial Intelligence in Education: Open Learning Environments: New Computational Technologies to Support Learning, Exploration and Collaboration, Frontiers in Artificial Intelligence and Applications, Vol. 50, eds. Susanne P. Lajoie and Martial Vivet, 750-752. Amsterdam: IOS Press, 1999.

[Weibel *et al.* 1998]

Weibel, S., J. Kunze, C. Lagoze and M. Wolf. "Dublin Core Metadata for Resource Discovery." Internet Engineering Task Force, RFC 2413. Internet URL: http://www.ietf.org/rfc/rfc2413.txt (September 1998).

Software Engineering and Design

[Hayes 1991]

Hayes, Barry. "Using Key Object Opportunism to Collect Old Objects." *ACM SIGPLAN Conference on Object Oriented Programming Systems, Languages and Applications*, 1991 (Phoenix, AZ, USA: ACM, 1991): 33-46.

[Hinostroza *et al.* 2000]

Hinostroza, Enrique, Lucio E. Rehbein, Harvey Mellar, and Christina Preston. "Developing Education Software: a Professional Tool Perspective." *Education and Information Technologies* 5:2 (2000): 103-117.

[Kam-Wah Wu 1993]

Kam-Wah Wu, Albert. "Paradigms for ITS (Intelligent Tutoring System)." *IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering, 1993*. Vol. 1 (Beijing: IEEE, 1993): 96-99.

[Keeling 1999]

Keeling, Harry N. "A Methodology for Building Intelligent Educational Agents." In *Artificial Intelligence in Education: Open Learning Environments: New Computational Technologies to Support Learning, Exploration and Collaboration*, Frontiers in Artificial Intelligence and Applications, Vol. 50, eds. Susanne P. Lajoie and Martial Vivet, 46-53. Amsterdam: IOS Press, 1999.

[Shih *et al.* 2001]

Shih, Timothy K., Shi-Kuo Chang, Jeffrey Tsai, Jianhua Ma, and Runhe Huang. "Supporting Well-Engineered Web Documentation Development – a Multimedia Software Engineering Approach toward Virtual University Courseware Designs." *Annals of Software Engineering* 12 (2001): 139-165.

[Virvou and Tsiriga 2001]

Virvou, Maria, and Victoria Tsiriga. "An Object-Oriented Software Life Cycle of an Intelligent Tutoring System." *Journal of Computer Assisted Learning* 17 (2001): 200-205.

Stochastic ITS models

[Boutilier 1999]

Boutilier, Craig. "Knowledge Representation for Stochastic Decision Processes." In *Artificial Intelligence Today: Recent Trends and Developments*, eds. Michael J. Wooldridge and Manuela Veloso, 111-152. Lecture Notes in Computer Science. Lecture Notes in Artificial Intelligence; 1600. Berlin: Springer, 1999.

[Branch *et al.* 1999]

Branch, Philip, Greg Egan, and Bruce Tonkin. "Modeling Interactive Behaviour of a Video Based Multimedia System." *IEEE International Conference on Communications, 1999*. Vol. 2 (Vancouver, Canada: 1999): 978-982.

[Gorniak and Poole 2000]

Gorniak, Peter J., and David Poole. "Building a Stochastic Dynamic Model of Application Use." *Sixteenth Conference on Uncertainty in Artificial Intelligence* (Stanford, CA: UAI, July 2000): 230-237.

[Jung-Jin Lee and McCartney 1998]

Jung-Jin Lee and Robert McCartney. "Partial Plan Recognition with Incomplete Information." *IEEE International Conference on Multi Agent Systems, 1998* (Paris: IEEE, 1998): 445-446.

[Mayo and Mitrovic 2000]

Mayo, Michael, and Antonija Mitrovic. "Using a Probabilistic Student Model to Control Problem Difficulty." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 524-533. Berlin: Springer, 2000.

[Millán *et al.* 2000]

Millán, Eva, José Luis Pérez-de-la-Cruz, and Eva Suárez. "Adaptive Bayesian Networks for Multilevel Student Modelling." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 534-543. Berlin: Springer, 2000.

[Murray 1999]

Murray, William R. "An Easily Implemented, Linear-time Algorithm for Bayesian Student Modeling in Multi-level Trees." In Artificial Intelligence in Education: Open Learning Environments: New Computational Technologies to Support Learning, Exploration and Collaboration, Frontiers in Artificial Intelligence and Applications, Vol. 50, eds. Susanne P. Lajoie and Martial Vivet, 413-420. Amsterdam: IOS Press, 1999.

[Rabiner 1989]

Rabiner, Lawrence R. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *Proceedings of the IEEE*, 77 no. 2 (February 1989): 257-286.

[Ueno 2001]

Ueno, Maomi. "Student Models Construction by Using Information Criteria." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 331-334.

[Vasileva *et al.* 2001]

Vasileva, Tatjana, Vladimir Trajkovic, and Danco Davcev. "Experimental Data About Knowledge Evaluation in a Distance Learning System." *Joint 9th World Congress and 20th NAFIPS International Conference, 2001*. Vol. 2 (Vancouver, Canada: IEEE, 2001): 791-796.

User Interfaces for Agent-Based Systems

[Elliott *et al.* 1999]

Elliott, Clark, Jeff Rickel, and James Lester. "Lifelike Pedagogical Agents and Affective Computing: An Exploratory Synthesis." In *Artificial Intelligence Today: Recent Trends and Developments*, eds. Michael J. Wooldridge and Manuela Veloso, 195-211. Lecture Notes in Computer Science. Lecture Notes in Artificial Intelligence; 1600. Berlin: Springer, 1999.

[Huhns and Singh 1998]

Huhns, Michael N., and Munindar P. Singh. "Anthropoid Agents." *IEEE Internet Computing* 2, no. 1 (January-February 1998) 94-95.

[Mitrovic and Suraweera 2000]

Mitrovic, Antonija, and Pramuditha Suraweera. "Evaluating an Animated Pedagogical Agent." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 73-82. Berlin: Springer, 2000.

[Moreno 2001]

Moreno, Roxana. "Contributions to Learning in an Agent-Based Multimedia Environment: A Methods-Media Distinction." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 464-465.

[Nijholt 2001]

Nijholt, A. "Agents, Believability and Embodiment in Advanced Learning Environments: Introduction to a Panel Discussion." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 457-459.

[Trower 1999]

Trower, Tandy. "Microsoft Agent, A Multi-Client Programming Interface for Interactive Computer Characters." In *Intelligent Agent Technology: Systems, Methodologies, and Tools*, eds. Jiming Liu and Ning Zhong, 379-383. Singapore: World Scientific, 1999.

Web-Based Intelligent Tutoring Systems

[Beaufils 2000]

Beaufils, Alain. "Tools and Strategies for Searching in Hypermedia Environments." *Journal of Computer Assisted Learning* 16 (2000): 114-124.

[Brusilovsky 2000]

Brusilovsky, Peter. "Adaptive Hypermedia: From Intelligent Tutoring Systems to Web-Based Education." In *Intelligent Tutoring Systems*, Proceedings of the Fifth International Conference, ITS, Montréal, Canada: June 2000, Lecture Notes in Computer Science; 1839, eds. Gilles Gauthier, Claude Frasson, and Kurt VanLehn, 1-7. Berlin: Springer, 2000.

[Larson and Koller 2001]

Larson, Richard, and Laura Koller. "Asynchronous Online Tutoring: Using Multimedia to Teach Difficult Concepts." Presented at the *Seventh Sloan-C International Conference on Online Learning: Emerging Standards of Excellence in Asynchronous Learning Networks*. (Orlando, FL, USA: ALN, November 16-18, 2001): Internet URL: http://reach.ucf.edu/alnconf2001/presentations/1373.ppt.

[Li Xiao *et al.* 2001]

Li Xiao, Li Jianguo, and Zhang Xiaozhen. "The Tuneup and Integration of Resources in Web-Based Learning." *IEEE International Conference on Advanced Learning Technologies, 2001* (Madison, WI, USA: IEEE, 2001): 391-394.

[Özdemir and Alpaslan 2000]

Özdemir, Bülent, and Ferda N. Alpaslan. "An Intelligent Tutoring System for Student Guidance in Web-Based Courses." *Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies, 2000.* Vol. 2 (Brighton, UK: IEEE, 2000): 835-839.

Web-Based Pedagogy and its Cognitive Impacts

[Abbey 2000]

Abbey, Beverly, ed. *Instructional and Cognitive Impacts of Web-Based Education.* Hershey, PA: Idea Group Publishing, 2000.

[Bates 2000]

Bates, A.W. *Managing Technological Change: Strategies for Academic Leaders.* San Francisco, CA: Jossey Bass, 2000.

[Cole 2000]

Cole, Robert A., ed. *Issues in Web-Based Pedagogy: A Critical Primer.* Westport, CT: Greenwood Press, 2000.

[diSessa 1990]

diSessa, Andrea. "A Principled Design for an Integrated Computational Environment." *Toward a Scientific Practice of Science Education.* Hillsdale, NJ: Lawrence Erlbaum Associates, 1990.

[Lipson 2001]

Lipson, Alberta. "A Three-School Comparative Analysis of Student Usage Patterns and Attitudes toward PIVoT." MIT Teaching and Learning Laboratory. Internet URL: http://www-caes.mit.edu/research/pivot/PIVOT_report1.pdf (July 2001).

[Maddux and Cummings 2000]

Maddux, Cleborne D., and Rhoda Cummings. "Developing Web Pages as Supplements to Traditional Courses." In *Instructional and Cognitive Impacts of Web-Based Education*, ed. Beverly Abbey, 147-155. Hershey, PA: Idea Group Publishing, 2000.

[Miller and Miller 2000]

Miller, Susan M., and Kenneth L. Miller. "Theoretical and Practical Considerations in the Design of Web-Based Instruction." In *Instructional and Cognitive Impacts of Web-Based Education*, ed. Beverly Abbey, 156-177. Hershey, PA: Idea Group Publishing, 2000.

[O'Sullivan 2000]

O'Sullivan, Patrick B. "Communication Technologies in an Educational Environment: Lessons from a Historical Perspective." In *Issues in Web-Based Pedagogy: A Critical Primer*, ed. Robert A. Cole, 49-64. Westport, CT: Greenwood Press, 2000.

[Okamoto *et al.* 2001]

Okamoto, Toshio, Alexandra I. Cristea, and Mizue Kayama. "Future Integrated Learning Environments with Multimedia." *Journal of Computer Assisted Learning* 17 (2001): 4-12.

[SchWeber 2000]

SchWeber, Claudine. "The 'Time' Factor in On-line Teaching: Implications for Faculty and Their Universities." In *Issues in Web-Based Pedagogy: A Critical Primer*, ed. Robert A. Cole, 227-236. Westport, CT: Greenwood Press, 2000.

[Vogel and Klassen 2001]

Vogel, Douglas R., and Johanna Klassen. "Technology-Supported Learning: Status, Issues, and Trends." *Journal of Computer Assisted Learning* 17 (2001): 104-114.