The Computational and Educational Viability of Deploying Intelligent Tutoring Systems

by


A dissertation submitted to the School of Computing in partial fulfilment of the requirements for the degree of Bachelor of Computing with Honours

University of Tasmania

(November, 2003).
TABLE OF CONTENTS

1. LITERATURE REVIEW ........................................................................................................ 4
   i. Introduction ....................................................................................................................... 4
      a. Evolution of the ITS ....................................................................................................... 5
   ii. Educational Underpinnings ............................................................................................ 12
      a. Piaget and Vygotsky ...................................................................................................... 13
      b. General Learning Strategies ....................................................................................... 16
   iii. Structure of typical ITS ............................................................................................... 17
      a. Pedagogic module .......................................................................................................... 18
      b. Knowledge domain module ........................................................................................ 23
      c. Student model ............................................................................................................... 24
      d. Designer interface ......................................................................................................... 27
      e. Student interface .......................................................................................................... 31
   iv. Delivery of Software ...................................................................................................... 32
      a. Traditional Web-Based, 3-Tier Architecture ................................................................. 33
      b. Agent distributed .......................................................................................................... 34
      c. Stand-Alone Applications ............................................................................................ 35
   v. Review implications ....................................................................................................... 36

2. PERFORMANCE: SURVEY & STUDIES ..................................................................... 37
   i. Questionnaire design and administration ...................................................................... 37
   ii. design of questions ......................................................................................................... 38
      a. Indicator of Performance ............................................................................................... 38
      b. Variables ....................................................................................................................... 41
      c. Additional Considerations .......................................................................................... 42
   iii. Observations ................................................................................................................ 43
      a. Resultant clusters ......................................................................................................... 44
      b. Conclusions about survey questions and methodology ............................................... 46
   iv. Additional Performance Components .......................................................................... 46

3. PROPOSED ARCHITECTURE ...................................................................................... 49
   i. Context of Architecture .................................................................................................. 49
      Centralised Institutions .................................................................................................. 49
   ii. Description: Components and Processes .................................................................... 51
      a. Web Service Yellow Pages ......................................................................................... 51
      b. Broker Agent ............................................................................................................... 51
      c. User Agent .................................................................................................................. 52
      d. Web Start and JADE ................................................................................................... 52
   iii. Example Tutor ............................................................................................................. 53
      a. Role ............................................................................................................................. 53
      b. Educational approach ................................................................................................. 53
      c. Interface ...................................................................................................................... 54
d. Knowledge Domain ........................................................................................................................................ 55

e. Pedagogy ....................................................................................................................................................... 56

f. Student Model ............................................................................................................................................... 57

4. DISTRIBUTION IMPLEMENTATION OUTCOMES .................................................. 58

 a. The Agent Context ....................................................................................................................................... 58

 b. The Student Model ..................................................................................................................................... 59

 c. Student_Agent ............................................................................................................................................. 60

 d. Broker_Agent .............................................................................................................................................. 60

 e. User Access to Tutor .................................................................................................................................. 61

 f. Summation .................................................................................................................................................... 62

5. CONCLUSION AND FURTHER RESEARCH ......................................................... 64

6. REFERENCES ............................................................................................................................................... 1

7. APPENDICES .............................................................................................................................................. 8

i. Appendix A .................................................................................................................................................. 8

 a. ITS Author Comments: ................................................................................................................................. 8

ii. Appendix B .................................................................................................................................................. 10

 a. Performance Survey Results: ...................................................................................................................... 10

iii. Appendix C ................................................................................................................................................. 13

 a. Student_Agent State Change Diagram ......................................................................................................... 13

 b. Broker_Agent State Change Diagram ........................................................................................................ 13

iv. Appendix D .................................................................................................................................................... 14

 a. Downloading the Algorithm Tutor Using Web Start .................................................................................. 14

v. Appendix E ................................................................................................................................................... 15

 a. Response.html: the result returned by the Broker_Agent........................................................................... 15

vi. Appendix F .................................................................................................................................................... 17

 a. Example DTD for Algorithm Tutor ............................................................................................................... 17

8. GLOSSARY OF ACRONYMS .................................................................................. 18
Overview of Thesis
This paper first presents a review of the literature that has been published on adaptive educational software. After briefly describing the evolution of these programs, the review covers the main educational and computational features of current systems.

The next section of the thesis discusses the survey conducted in order to ascertain information about the performance of Intelligent Tutoring Systems. The results of this survey are used to inform the following section.

After the survey, the paper outlines a proposed architecture for the decentralised distribution of adaptive educational software and discusses the issues surrounding the architecture. The results from implementing this architecture follow in the next section.

Conclusions and possible further research are presented last to complete the work. Key results and have been appended as well as a glossary of acronyms.

Literature Review

Introduction
While the value of human interaction to the education process is unquestionable, there are instances where people either choose not to be involved in face to face human contact or are denied access to it. People with disabilities – mental, physical and social - often either cannot attend classes or would prefer not to be involved in human interaction, but are still motivated to learn. Personal experience has shown that a portion of teenagers find the structures of some school environments offensive (for a variety of reasons) and are quickly distracted from learning. People in wheelchairs might find it too much effort to cope with the travel. Educational institutions may not be adequately resourced to provide the staffing levels needed for personal attention. Timetables and locations may be difficult to co-ordinate. Further, users independent of teaching institutions such as Medical Practitioners may wish to update specific skills and knowledge. There are many situations where computer-based tutors might be preferred. As well there are educational applications that do not need the presence of a human teacher, especially if the software adapts to student performance. Practice is usually a very individual process requiring self dialogue and
evaluation. Modern Intelligent Tutoring Systems are quite capable of providing this sort of service.

**Evolution of the ITS**

The ITS emerged from expert system developments in Artificial Intelligence. Diagnostic software was able to make decisions and justify the reasoning behind those decisions. The process was based on the ability of the software to classify data according to production rules. At the same time there were a number of educational tools being developed, like spelling games and arithmetic testers. The combination of production rule Artificial Intelligence with educational applications led to educational software that adapted its student interface and content to the individual student. The adaptation was dependent upon which rules were satisfied.

**Intelligent Tutoring Systems**

Intelligent Tutoring Systems are the result of the specialization of AI-based educational software into individual interactive components. Components include the knowledge domain covered by the tutor, a model of the student and a set of “rules” for deciding upon the pedagogy presented to the student. (Pedagogy would include the content and sequence of the material plus the type and content of the feedback given to the student.) The components can be constructed in a variety of ways, including rule sets, concept-relationship graphs and Object Oriented frames.

“Intelligent Tutoring Systems (ITSs) are computer-based instructional systems that have separate data bases, or knowledge bases, for instructional content (specifying what to teach), and for teaching strategies (specifying how to teach), and attempt to use inferences about a student’s mastery of topics to dynamically adapt instruction.”(Murray, 1998).

![Diagram of Intelligent Tutoring System](image)

**Figure 1.** A generalized architecture of an Intelligent Tutoring System, showing typical interaction paths between the three models and the user interface.
An extra layer of Artificial Intelligence such as a neural network, fuzzy or Bayesian logic, or a genetic algorithm is often added to parts of an ITS to establish weighting for decisions – to prioritize them. These mechanisms are typically used to reduce the problem space and to more efficiently utilize the ITS components. (Murray 1999). Murray notes the following benefits of an ITS compared to standard commercial tools for computer assisted instruction:

- Content is distinct from strategies
- Instructional content is modularized for re-use
- Generic teaching strategies can be used with different content
- Abstract knowledge structures can be used as the basis of presentation rather than making decisions at the content level
- Tutor behaviour can be easily modified – a single change to the definition of a strategy can apply to the whole curriculum
- Module reusability allows usage for several purposes – e.g. a topic can be used for information, examples, summary and assessment
- Student learning preferences and needs are more flexibly met
- Definition and modification of instructional content is easier for the Instructor (Murray, 1998).

**Adaptive Hypermedia**
Adaptive Hypermedia have emerged in a similar manner to ITSs, but from multimedia and Web-based roots. Initially Artificial Intelligence was an add on to hyperlinked media, rather than central to the operation of the software. The intelligence commonly determined which hyperlinks to show and in what sequence to present the content, usually based on some intelligent model of the user. (Brusilovsky 2000). When used for educational purposes, any distinction between ITSs and Adaptive Hypermedia starts to blur.

**Authoring Tools**
Authoring Tools have been created which enable instructors to create ITSs through an interface built on generalized concepts of teaching strategy and domain knowledge. Preliminary training is needed, but the level of expertise needed to construct a tutor is vastly reduced. Figures on construction time indicate that preparing an ITS is on a par with the standard Computer Assisted Instruction (CAI)
packages available commercially. Efforts by Murray and Woolf reported in 1992 showed that it took 100 person hours to develop one hour of instruction, compared with 100-300 hours for non-intelligent CAI. Eon and REDEEM, two authoring tools, are currently being used in a number of research areas and have been evaluated extensively. REDEEM takes 90 minutes of training to operate. By using content that has already been prepared for CAI, one hour of instruction can be produced in about three hours. There is a trade-off between the ease of use of the authoring tool and the resulting tutor’s sensitivity (the depth of knowledge that can be expressed and its adaptability to student needs). (Ainsworth et al. 1999, Murray 1999). Three authoring tools REDEEM, EON and the PACT tools are described more fully below in the discussion on ITS components. Any ITS that has a designer interface is to some degree an authoring tool.

The blend of Intelligent Tutoring Systems and Adaptive Hypermedia has evolved into Adaptive Web Based Education Systems. (Brusilovsky 2003). It is now possible to redefine the once discrete intelligent tutor. The components of a tutor can belong to more than one system. If a particular component is network-based, then other systems can share it. Further, network-based ad-hoc systems can form and dissolve dynamically.

Such dynamics have occurred in the area of e-commerce already – with vendors and customers using agent technologies to generate the most favourable allegiances at the time. Examples of systems include AuctionBot, MAGNET, eNAs, Casba and MARI. Significant research into these areas is being carried out in Europe, the US, Australia and Japan. (Kowalczyk et al. 2002),

**Intelligent Agents**

Intelligent Agents when applied to education can implement a distributed conceptualization of an ITS. The components of the ITS can be seen as individual entities able to communicate with each other. Advantages of this approach include the ability to provide flexibility for data sharing and the ability for each component to behave both autonomously and in a collaborative manner. Agents have another advantage – mobility. Applications can be created where discrete units migrate to the end-user’s machine and communicate with other agents on the network. When one
explores the idea of individual components of an ITS free to collaborate and locate themselves, seemingly endless possibilities spring to mind. More than one “tutor” might be able to share the data of one student’s agent, for example, reversing the usual construct of one tutor for a group of students. Further, the tutor designed for a particular course at one university might become part of a different course altogether – at a different university.

There are a number of shifts in perspective being theorized as a result of the fluid nature of the Internet. One is the notion of student centred learning rather than the traditional instructor centred approach. The student rather than the instructor is seen as choosing the subject matter, the content and the media through which learning takes place. The student may opt for an instructor as part of the learning experience. In the traditional perspective the learning, subject matter, materials and media are controlled by the instructor once the student has enrolled. (Graves 2001), Research into the implementation of this shift in viewpoint has begun. As one instance, the computing department of Open University, UK called for PhD students in the area of student-centred e-learning design in October 2002. (Minocha and Rapanotti 2002),

**Standards**
Sharing implies standards. Standards for describing e-learning materials are emerging. The IEEE has created a draft standard for Learning Object Metadata (LOM), (IEEE 2002). These have been implemented by IMSGlobal in an attempt to describe and classify learning objects and thus make them universally accessible. (IMSGlobal 2003),

The Semantic Web (Berners-Lee 1998) is a vision of unifying the World Wide Web by giving meaning to its contents. Searches of the Web, currently by keyword, will be able to be carried out by the concept or meaning behind the keyword instead. This meaning is expressed as a variety of unified knowledge representations or ontologies. The representations can be identified with a URI (Uniform Resource Identifier). This will allow knowledge to be stored and created, searched and matched freely by intelligent agents. (Berners-Lee et al. 2001), Knowledge representations are fundamental to Intelligent Tutoring Systems.
If the description of the tutor’s knowledge domain is archived, then many independent tutors could be created by different institutions using the same knowledge representation. These tutors would be intelligent exercises or ‘pedagogic agents’. Institutions would be able to share these agents when teaching a particular topic (from the current instructor centred perspective). From the student’s perspective, there could be a choice of pedagogic agents (and thus a choice of strategies) to explain a given concept. The student’s knowledge could be broadened if the student’s agent could retrieve “tutors” that service similar domains, or locate a related domain that the student may not be aware even exists.

One way to view the ITS with respect to the semantic web concept is to draw an analogy between a text book and the Dewey or Library of Congress Classification Scheme. The instructor might select the textbook or the independent student might select the textbook that best suits his or her learning style and level, or the librarian – as an agent of either party - might provide a list of recommendations.

Agent recommendations would include where to find the tutor on the shelves (URI), an indication of the body of knowledge referenced by the tutor (another locator or perhaps an entire knowledge representation) and other LOM standard data such as author and title, running platform and size.

There is great potential for enhancement of the educational experience through the rebuilding of ITS components in the new technological framework.

Intelligent agents are currently being developed in Brazil for use in the distance education context. A communications agent is used to manage interaction between agents, a group of pedagogic agents are in charge of learning activities and strategies and a student agent is responsible for user modeling and agents’ coordination. (Silviera and Vicari 2002),

Collaboration is possible through the use of agents. Collaboration occurs between agents and between students, creating a “society”. This extends to collaborative learning between students – allowing students to be involved in remote group work. Students can discuss and share common interests. (Shang et al. 2001),
This thesis will outline a practical agent architecture for deploying intelligent tutoring. It must be extensible and be flexible enough to expand with the changes preempted above. To do this requires the decentralisation of some of the structures commonly kept on a server. Some types of student models need to be generated and stored on the student’s machine, in particular the models that describe the educational level, subject matter and topics that locate the student.

Much research has gone into just how agents communicate with each other. Standards are emerging for agents, the language they use to communicate, their behaviours and the language used to describe the knowledge that they share.

The JADE project (Bellifemine and Trucco 2003), uses distributed environments based on Java RMI. The intelligent tutor basic components can be integrated with intelligent agents that communicate through ACL (Agent Communication Language), an emerging standard (FIPA 2003).

Accompanying the shift towards a dynamic and fluid Internet, relevant standards have been evolving, including the FIPA (Foundation for Intelligent Physical Agents) standards for intelligent agents and agent communication and the Object Management Groups standards for object exchange (OMG) (OMG 2003).

“Because it is likely that agent technology and object technology specifications will eventually overlap (e.g., both need namespaces, have lifecycle services, use persistence, address mobility, etc.) then it is desirable that the two families of specifications not conflict needlessly.

FIPA’s new agent reference architecture will develop well-defined abstract architectures and map these to concrete instantiations. OMG specifications may provide a good basis for these instantiations, providing services such as naming, object transport and messaging. In addition, there may be agent specific services that OMG must support. It is important to co-ordinate these efforts in order to ensure that there is a good synergy between the two specification efforts.” (OMG-FIPA 1999).
The JADE project is well documented and implements these standards using Java technology.

XML web services developments are also intricately tied into the new Internet distributed programming technologies – being a system of exchanging objects in XML-defined text. XML-defined text underlies the standards set for agent communication. The similarity between the two perspectives on distributed networking suggests some convergence in the future.

Two web services standards groups are emerging here. The open source group is aligned with OMG which have developed CORBA (Common Object Request Broker Architecture) object exchange and WSDL (Web Services Definition Language) for implementations of web services and object exchange. OMG enters web services from the software design/modeling perspective. The Microsoft-IBM group, among other business focused groups, is trying to make faster progress and is implementing standards set by the Organization for the Advancement of Structured Information Standards (OASIS). These standards are very business specific and are being delivered faster than the OMG standards. In particular OASIS has just delivered the specifications for security - Service Provisioning Markup Language (SPML). (Taft 2002, McCarthy 2003).

OMG wish to introduce Enterprise Collaboration Architecture (ECA). The following quote from InfoWorld hints at the cause of some of the underlying divisions, with the focus on mapping rather than evolutionary products as the source of standards:

“By defining and storing business collaborations as models, companies are provided immunity to the ongoing evolutions of underlying middleware, data formats, and other infrastructure, said Jon Siegel, vice president of technology transfer at OMG, based in Needham, Mass.” (April 2002).

The agent approach seems to be a more stable and individualised conceptualisation and a better choice for pursuing student-centred goals.
**Current research**

Current research on ITS is focusing on the development and evaluation of educational strategies implemented by intelligent tutors (some research is listed in the pedagogic module description below). Research is also being conducted on standardized ways of modeling users and knowledge, of intelligent document retrieval and refining existing computer – based training delivery systems like WebCT. (Tang and McCalla 2003, Mazza and Dimitrova 2003, Kosba et al. 2003).

The complexity and usefulness of ITS is resulting in worldwide collaboration between researchers. Papers frequently have three and four authors, from different institutions. For example, there is collaboration between Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM) in France and the Rumanian Academy Research Centre for Machine Learning, Computational Linguistics and Conceptual Modelling in Bucharest. They have collaborated to develop a tutor EU INCO that teaches the (English) language of finance as a foreign language. (The tutor features an agent that searches the Web looking for metaphors to use in teaching linguistic concepts) (Trausan-Matu et al. 2002).

An editor of a definitive conference dealing with ITS research comments in the introduction on the selection process for research papers:

“Toy systems were criticized, real scale, well evaluated experiences were encouraged. Real size research on ITS favors real size research groups. As a result many papers are signed by multiple authors. Geographically isolated scientists tend more and more to collaborate remotely via Web…..” (Cerri et al. 2002).

Many teaching strategies have been trialed in the history of ITS development. These strategies, their success and the educational theory upon which these strategies are based are outlined next.

**Educational Underpinnings**

ITSs have been demonstrated to be successful as educational tools for the last 15 years. Lesgold (1990) is cited by Murray as revealing students working with an Air Force electronics trouble shooting tutor for 20 hours and finishing with the
equivalent of 40 months (almost 4 years) of on-the-job experience. In another example, students using Anderson’s LISP tutor (1990) completed programming exercises in 30% less time than those receiving traditional classroom instruction and scored 43% higher on the final exam. (Murray 1998). Now “we can reliably build intelligent tutors that are about half as effective as human tutors.” (Corbett et al. 2000). It is clear that one tutor per two students is more than most institutions can afford and that intelligent tutors will only become more common. As research continues, the effectiveness of ITSs will also improve.

ITSs generally use an underlying learning theory to underpin the teaching strategies (pedagogy) employed in the tutoring process. Mostly, detailed educational paradigms are omitted in favour of recognized, general, learning strategies. These are used to justify the specific strategies employed by specific tutors. The strategies are very much dependent on the purpose of the tutor – teaching the language of the marketplace to under-graduates, primary geometry, teaching future pilots to fly or students with disabilities that cannot sequence material for themselves. The major streams of thought are introduced next, followed by general strategies, with specific strategies handled as part of the discussion on the pedagogic model below.

**Piaget and Vygotsky**

Not surprisingly, most of the theories referenced by the authors of ITSs are based on the constructivist models of cognition developed by Piaget and Vygotsky. Learners build their understanding and concepts rather through engagement than as empty vessels into which knowledge is poured. To over-simplify, Piaget developed his theories of innate learning processes based on the way humans learn to manipulate the natural world around them. Vygotsky was inclined to see the process as one that involved the society around a person – that there were significant cultural determinants in the process. His views are used to understand the ability to learn technologies and apply them in different ways, to underpin the importance of words and verbalising, and to provide educational “scaffolding” when the learner was exploring his/her “Zone of Proximal Development”.

“Zone of proximal development is the difference between the child’s capacity to solve problems on his own, and his capacity to solve them with assistance. In other words, the
actual developmental level refers to all the functions and activities that a child can perform on his own, independently without the help of anyone else. On the other hand, the zone of proximal development includes all the functions and activities that a child or a learner can perform only with the assistance of someone else. The person in this scaffolding process, providing non-intrusive intervention, could be an adult (parent, teacher, caretaker, language instructor) or another peer who has already mastered that particular function.” (Schütz 2002).

Beyond this zone is the realm of cognitive overload – where more information and decision-making is present than the learner can cope with. Authors of ITSs have been mindful of this situation and usually adapt the user interface to the user’s level of performance. (Aleven 2002, Brusilovsky 1997).

**Cognitive/constructivist**

Cognitive tutors have grown from work by Anderson, Corbett and others at Carnegie Mellon University. (Anderson et al. 1995). The basis of their tutors is the ability of the tutor to solve the same problems that the student is asked to solve. The student’s ability is traced as a subset of the knowledge domain in the tutor. The tutors rely on the ACT-R theory of skill knowledge. Here knowledge is assumed to be either declarative or procedural – either items (‘chunks’) of knowledge, or skills. Both are developed as a result of practice. Goals are set (problems to be solved) and student solutions compared to the solution paths predicted by an expert. The expert may predict commonly made mistakes as well. The steps on the path are represented by production rules. The mental processes modeled as productions (skills) are “matching”, “selection” and “execution”.

The Carnegie-Mellon team has been using the tutors to further study the psychology of cognition. Some findings include:

- US algebra students prefer mathematics problems in words rather than algebraic expressions and this is not reflected in Russia, where algebra starts in grade 2
- Many students lack the metacognitive skills to seek help when they need it
- Greater learning and transfer occurs if students are asked to explain their solutions (Koedinger 2000).
Continuous evaluation is the key to improving the quality of intelligent tutors. The findings have ramifications for human tutors as well.

**Active Learning / dialogue**

Active Learning is a concept that has been used for centuries in the practice of teaching. There have been a number of attempts to formalise a definition of the processes involved in the many successful teaching strategies encompassed by the concept.

Modell and Michael (1993) are quoted as defining “an active learning environment as one in which students individually are encouraged to engage in the process of building their own mental models from the information they are acquiring. In addition, as part of the active learning process, the student should constantly test the validity of the model being constructed.” (TRC 1999).

This is similar to the constructivist model above and to theories of metacognition often used to teach “real world” problem solving to adult learners. In the latter instance, metacognition is really developing the model of one’s own learning, and continually testing that model. (McLoughlin and Hollingworth 2001).

Active learning occurs through active involvement with instructional material either through **collaborative or self dialogue**. The major “Active” experiences are **observing** and **doing**. (Shang et al. 2001). When a student studies textbooks or worked out examples they learn with greater understanding if they explain the material to themselves. However, not all students self explain or do so thoroughly. (Renkl et al. 1998). Much of the challenge with ITS pedagogic strategies is to create strategies that compel or coax students to be engaged in some form of self explanation – drawing analogies or understanding metaphors, inferring, concluding, rehearsing, summarizing – in some type of medium - written, graphical, auditory or spoken.

**Emotional State**

Recent study into the emotional state of learners focuses on the idea that the degree and type of external knowledge presented to the student should be based on the emotional state of the learner at the time, for the best learning outcomes to occur. Kort, et al address issues such as cognitive overload, frustration, awe, hopefulness, confusion and curiosity as having bearing on the receptiveness of the learner. (Kort et al. 2001).
Research at the Federal University of Rio Grande in Brazil models the mental states of students. Concepts such as effort, confidence and independence are used as guides to the teaching/learning strategies employed. (Vicari 2002).

**General Learning Strategies**

**Drill, Exercises and Practice**
The aim of most tutors is to provide practice through repetition. This is an age-old approach to learning with the aim of helping memorize skills and knowledge.

**Learning by analogy or metaphor**
Analogy is treated by Meyer as “similitude between elements from semantically separate knowledge domains.” (Meyer 2002). She quotes Sticht and Ortony as educational theorists who see analogy as an important learning tool. She argues that the reason that this type of learning has not been widely used in ITS to date is because of the complexity of modeling knowledge domains. Modeling analogous domains as well as the domain to be taught is economically prohibitive and potentially computationally expensive.

An application was created to teach network routing using analogies such as post office delivery of mail, road networks and rail networks. A concept map was drawn and analogous tags (using words from the analogous knowledge domain) were added to the diagram. The analogy tags were well received by the students, who wanted to be able to create their own analogies. (Meyer 2002).

**Case-based learning – learning by example**
The principle used in learning by example is that the study of cases and worked examples gives the student the opportunity to understand the underlying concepts. Underlying concepts when applied to the domain of mathematics include the use of operators that apply to a class of problems, the transformations that occur when these operators are used and the sequence in which the operators should be used. (Chi et al. 1989, Anderson et al. 2002).

Mathan and Koedinger cite recent work by Renkl, Atkinson and Maier (2000), which indicates that gradual introduction of problem solving into the worked examples, enables even better learning by example. Partially worked examples focus the
student’s attention on the crucial parts of the problem and allow the student to search for knowledge of sub goals as well as giving the opportunity for self evaluation. The risk of learning underlying concepts, as well as the simpler operations demonstrated by the examples, is that the operator technique may not be learnt as thoroughly. The operator techniques may be obscured by cognitive overload from the additional (underlying) concepts. (Mathan and Koedinger 2002).

**Open Student Model Learning**
In this strategy, the student is informed of the way the instructor is viewing the student’s knowledge, skills and progress. The intention here is to engage the student in thinking about their own learning and at the same time to ensure that the model of the student (perhaps an affective description not just assessment) is as accurate as possible. (Zapata-Rivera and Greer 2002). There are also political implications to gaining the student’s tacit consent. Judgment is not made about a person behind their back. The educational underpinning is again this notion of active learning and dialogue with self. “Students who engage on the meta-cognitive level achieve significantly better results than those who do not.” (Hartley and Mitrovic 2002).

**Structure of typical ITS**
The architecture of Intelligent Tutoring Systems is varied and complex when dealt with at the modular level. Unique modules are often created to deliver very task specific content. The delivery of the Dijkstra-Gries formal programming methodology ITS for example, is described in four layers identified by the service provided, the upper levels drawing on services from the lower levels. The lowest level has components *User Modelling, Advice* (syntax semantics, principles and strategies) and *Expresso* (an arithmetic and predicate logic toolbox). *User Modelling* comprises two sub modules – (the methodology and concept models in one, and the student’s history and knowledge model in the other). At the highest level one sub-module of the *Tutor* is the assistant. It draws on *Advice* and *User Modelling* as well as the *Calculator* (The *Calculator* is two levels below and draws on services from *Expresso*). (Ng et al. 1995). The architecture of the Dijkstra-Gries tutor is specialised to deal with the delivery of formal methods arithmetic and logic.
Research into refinements that introduce efficiencies can modify or add new modules. Additional functionality is added to most tutors as they mature. The best approach to describing the general architecture of an ITS is to describe the basic modelling that most ITS are required to perform. Most ITS model the student, the teaching technique and the structure of the knowledge to be taught.

There also needs to be an interface for the learner and, if able to be authored, an interface for the instructor.

**Pedagogic module**

The pedagogic model is the component of an ITS concerned with determining which strategies to employ given a specific state of the student model. It is usually a set of production rules or some alternate if/else structure that evaluates a state. A typical rule might be:

\[
\text{if}\_\text{state/implication } X :\text{then}\_\text{action/implication } Y.
\]

Just what sort of content (representation of a concept) should be loaded, what level of feedback, when to provide feedback and what level of difficulty to use, are all decisions that might be taken by the pedagogic model. Different specific strategies are used depending upon the purpose of the ITS and the appropriate general educational research used. Generally the strategy rules are hard coded into the system by an expert in teaching the subject matter.

Rules can be built by neural networks and other machine learning techniques, if case-based data is available. The “neurule” is an interesting example and can aid in developing teaching heuristics. Neurules are a hybrid form of knowledge representation. Rules are supplemented by a “significance factor” or bias. The significance factors are developed using neural networks and associated with the existing symbolic rules. This bias is used to prioritize the firing order of the rules.

Neurules provide advantages over symbolic rules in speed, space and the ability to derive conclusions from partially known inputs. They provide advantages over neural networks in that they are easy to update without affecting existing knowledge and they provide natural explanations for their ‘reasoning’. (Prentzas et al. 2002).
Specific Strategies

**concept mapping**
Concept maps are intended to give the student an overview of the territory they are exploring, to save them perhaps erroneously inferring where they are in the knowledge domain. An interesting twist to this technique is StyLE-OLM which uses an interactive concept map of the student model (the perceived map of the student’s ability) as seen by the ITS and the instructor. StyLE-OLM allows the student to modify the map of learning that the ITS has formed, requiring that the student be involved in valuable meta-cognitive processes. (Dimitrova et al. 2002).

**providing research materials and selected, sequenced data**
EU INCO, mentioned earlier, is typical of most tutors in that it adapts the material presented to the student, to suit what needs to be learnt. By finding uses of financial terms on the web, the tutor is able to give the student a list of websites where a targeted word is used in context. (Trausan-Matu et al. 2002).

**exercises**
The standard teaching strategy of offering drill exercises in problem solving is one that takes a considerable amount of effort on the part of the instructor. Preparing online materials is often very time-consuming. Martin has added an automated problem generator to the SQL tutor. The new component takes advantage of the Constraint Based Modeling of the student (see below) that is used in all tutors from the University of Canterbury. The list of violated constraints (concepts not understood by the student) is used as the basis for generating new problems. Martin found that the use of the generated problem set improved students’ learning speed by a factor of two. This is attributed to two possible causes (or a combination, either more practice as a result of more exercises being available or better selection of the exercises generated appropriately for each student. (Martin and Mitrovic 2002).

**giving immediate/ belated feedback**
Experiments with KERMIT, the database design tutor, show conclusively that automated feedback during the tutoring process improves the performance of students. (Suraweera 2001). This supports findings by Corbett and Anderson, whose
results suggested that there was no difference with respect to when feedback was given, only whether it was given. (Corbett and Anderson 2001).

There is debate about when an ITS should provide feedback on student performance. Studies have shown immediate feedback (immediately a student makes a mistake) and belated feedback at the end of the exercise (Mitrovic 1998) result in similar short term performance in post tests. Indications are that long-term, deep understanding is best developed by leaving feedback until the student has had a chance to test their own actions. (Mathan and Koedinger 2002).

Mathan and Koedinger cite Schmidt’s guidance hypothesis as an attempt to explain what is happening. Feedback is seen to precisely direct the student’s actions after each feedback presentation. There is a short-term gain when performing on tasks that precisely fit the actions that have just been learnt. However feedback can mask the cues inherent in the natural task environment by training the students to be dependent on feedback. “Generative” skills, the ability to choose operators and apply these to specific task contexts, are handled well.

Instant feedback may prevent many skills such as error detection and metacognition from occurring. These “evaluative” skills may go unpracticed. (Mathan and Koedinger 2002).

**simulation / external representation**

Probably the best example of the effective use of simulation is in the training of pilots on a flight simulator. The advantage of having an intelligent, distributed simulation is that there is the opportunity for feedback and practice anywhere and at any time. (Minko et al. 2002).

Clearly simulations are very useful for teaching familiarity with specialized equipment and the procedures needed to operate them. They can however, be used to teach quite abstract skills, such as the problem solving needed in military tactics. Here troops and military equipment are potentially the basis for developing new procedures, not just existing ones. (Joab et al. 2002).

AMPS is being used to experiment on the teaching of analogy by comparing simulations of liquid and electrical current flows for physics students. Analogy is seen as a complex way of learning, dependent on a clear and complete understanding of a base knowledge domain before being able to comprehend a new, analogous domain. Brna and his associates are experimenting with the parallel development of
understanding of both domains. The problem of cognitive overload is one they are testing. Below is a snapshot of one simulation.

![Diagram of two analogous currents](image)

**Two analogous currents**

**Figure 2. Educational strategy using simulation.** (Brna and Duncan 1996).

The border between a simulation and an external representation (graphical, animated or whatever) is obscure and (in the context of this paper) irrelevant. Both terms are used freely in the literature. How would one categorize SimForest? SimForest is an ecological simulation which allows students to inspect and manipulate the underlying mathematical equations that influence ecological balance. (Murray et al. 2001).

**worked examples /cases**

Mathan and Koedinger experimented with an active walk-through of worked examples, where instead of the solution being passively presented (and students then being tested on their learning), the examples are presented with the student required to answer questions, with assistance, as the solutions are shown. As expected, those students who had the walk-through did better than those without. (Mathan and Koedinger 2002).

**multimedia / preferred medium presentation**

It is now well established that ‘a picture is worth a thousand words’ and that clear graphical representations are often easier to understand than verbal ones. This is a common teaching strategy and exploited at length in most refined ITS. A current example is e-KERMIT, where the system’s understanding of the student’s ability is displayed as a series of labeled sliders that represent each assessed criterion. Each slider shows the level of achievement as a proportion of the total possible.
natural language dialogue
A number of ITSs are used for the teaching of spoken languages and use natural language parsing units (NLUs) in their architecture (as the knowledge domain). A notable current use of an NLU is in the work of Aleven, et al. Their extension of Anderson’s geometry tutor requires the students to verbalize geometry theorems as a form of self explanation. It asks the students to construct, from a restricted vocabulary, explanations in their own words. The new architecture shows it is capable of this task and the results are encouraging, with room for further experiment and improvement. (Aleven et al. 2002).

Another implementation of dialogue is that of the two virtual assistants, the professor and the companion. When the learner should be asking for help but doesn’t, the SIMPA tutor gets the companion assistant to ask the professor assistant. The appropriate concept is introduced through the dialogue between the two assistants. (Duquesnouy et al. 2002).
**Knowledge domain module**
The knowledge domain is used for defining the content to be presented by the pedagogic model. It is also used for comparing with the student model to discover what the student does/doesn’t know. Often an Expert module is used to extract correct results from the knowledge domain.

The main concept involved in modeling any knowledge is the notion of ‘ontology’. Trausan-Matu quotes Gruber in defining an ontology, in the context of Artificial Intelligence as being “a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents”. (Trausan-Matu et al. 2002). An ontology is the set of concepts and relations that are considered fundamental to an area of knowledge and it is typically represented as either a semantic network (as is the case with Trausan-Matu et al’s ontology-based meta crawler) or as a hierarchic structure of possible pathways (as is the case with Mitrovic et al’s SQL tutor (Mitrovic 1998) and the Computer Intrusion Detection Tutor (Woo et al. 2002.).)

Nodes in the above structures are typically topics, concepts, skills, facts, misconceptions and sometimes exercises. The links between the nodes are usually labeled relationships. Frames are often used to represent nodes because of the ease with which they can be programmed in an object-oriented programming language. If a hierarchical vision of the knowledge is created, it frequently is used to represent levels of granularity like:

\[
\text{Course has Units has Lessons has Topics has Subjects has Exercises.}
\]

Trausan-Matu, et al take advantage of the less tightly defined structure of the semantic network to specialize their ITS by adding the ability to grow the ontology in an organic, holistic manner. (Trausan-Matu et al. 2002). Using one ontology of domain concepts and another of metaphors representing those concepts (a “corpus”), they add a component to their ITS which crawls the web looking for new metaphors to add to the corpus. (The EU INCO project is about learning foreign [English] finance terminology). The system uses XML databases to maintain the ontologies so
it could be argued that, because XML itself is based on a hierarchy of elements, there is in fact a hierarchical meta structure to the knowledge representation.

Functional languages like Prolog and LISP are also used to map knowledge, but these are usually in the form of rules that link variables. The model tracing tutors from Carnegie-Mellon move through lists of rules such as those shown in the section on designer interfaces (Figure 6) below.

**Student model**

The Student model contains a description of the state of the student with regard to his/her educationally significant attributes. Attributes can be as specific as short term knowledge and misunderstandings relative to a given topic, or can be more general descriptions of skills (strengths and weaknesses) as well as including an estimation of the general level of mastery of a given knowledge domain. Significantly, educational attributes can include quite personal, affective qualities such as the degree of motivation, dependence and confidence. (Suraweera 2001).

**Constraint Based Modeling (CBM)**

Constraint Based Modeling appears in a number of Intelligent tutors that have been created recently, including one for teaching context free grammar (Reyes and Galvey 2000), and three from the Department of Computer Science at the University of Canterbury in Christchurch, New Zealand (including the example of the SQL tutor, below). These models draw from concepts formulated by Stellan Ohlsson over a number of years, first published in 1992. (Ohlsson 1992).

Mitrovic argues against the use of the Cognitive approach:

“If the goal is to model student’s knowledge completely and precisely, student modelling is bound to be intractable.…. 
….however, a student model can be useful even if it is not complete and accurate …. 
….CBM focuses on faulty knowledge, realizing that it is not sufficient to describe what the student knows correctly. The basic assumption is that diagnostic information is not hidden in the sequence of student’s actions, but in the situation (or the problem state) that the student arrived at.” (Mitrovic et al. 2001).
CBM tests the final solution to see if it is in the knowledge domain rather than
testing each cognitive step on the way to that solution and thus only generates final
rather than immediate feedback. Recent results by Mathan and Koedinger (as already
mentioned) suggest this may be an advantage rather than a problem.

The SQL tutor uses CBM to model the student’s level of knowledge and compare
this to the knowledge base. It models the knowledge domain in the form:

“If <relevance condition> is true for the student’s solution,
THEN <satisfaction condition> must also be true.”

If the student’s response to a given problem is considered relevant to the problem
then the student’s solution is evaluated. Success leads to no further action, if the
student fails then feedback is given based on the error.

The SQL tutor builds a correct solution for the problem that the student has
attempted, but has answered incorrectly. It picks up the student’s approach, rather
than the preferred solution, by matching solution patterns with the student’s answer.
Feedback is derived from the outcome of the comparison.

The level of difficulty of the next problem chosen is determined by the number of
violations of particular constraints. An added layer of machine learning is used to
select a target problem that best teaches the constraint(s) deemed suitable (Martin
2001).

There are very few assumptions made about the students’ style of learning, the
motivation that they bring to the exercise, their degree of independence, etc. The
only real assumption is that the solution proposed by the student is ‘irrelevant’ to the
given subsection of the knowledge domain. It is judged that the student’s irrelevant
attempt cannot be correct.

If the response is irrelevant then the ITS cannot provide a strategy to help the student
to learn. How the tutor determines irrelevance must be based on the granularity of the
knowledge domain.
Overlay
Overlay modelling of students is based on comparison of student performance with an expert’s mastery of a knowledge domain. The domain is expressed in generic terms such as rules and concepts and the student’s mastery of these aspects of the domain is calculated. (Suraweera 2001).

Cognitive
The student modelling technique developed over the past two decades at Carnegie Mellon University for algebra, geometry and programming tutors is based on the cognitive learning theory previously mentioned. The approach, termed “model tracing” is top-down in nature. The knowledge modelling uses production rules to link “chunks” of knowledge to goal oriented (problem solving) decisions. All possible paths as well as errors (“buggy rules”) are mapped to the knowledge base. The student’s performance is traced as he/she descends the decision tree to a solution. Feedback is provided at the point of error and a long term record of student ability is also kept for providing further tasks.

The problem space is large and creating each production rule is laborious, but the tutors have been shown to be educationally successful. (Anderson, 1995) (Suraweera, 2001 pp.12-14).

Stereotyping
The “Fixed” stereotype approach to student modelling captures the student’s responses and then assigns the student a stereotype based on the responses. Students may be moved from one stereotype to another. For example the student may move between performance levels, but the descriptions of the levels remain fixed.

The student in the StyLE-OLM scientific terminology tutor of Dimitrova, et al, 1998 starts with an initial stereotype that is then modified by student responses. Overlay modelling is used to draw new conclusions about the student based on natural language dialogue. (Dimitrova and Dicheva 1998, Suraweera 2001).

Belief Network
The adaptive lesson delivery tool IDEAL models students using Bayesian probability. It infers the skill of the student from performance data expressed as a
belief network. For a given skill, whether the student is novice, intermediate, advanced or expert is determined by listing the likelihood of the student answering certain categories of questions correctly. To reduce the size of the calculation for purposes of efficiency, the number of probabilities is based on categories of questions rather than on individual questions.

The probability for a given skill level is calculated as the total of the probabilities for the prior (prerequisite) skill levels plus the probability that some of the errors were slips plus the likelihood that some are lucky guesses.

Shang states that the advantages of this modelling approach are:

- Questions can be added dropped or moved between categories with minimal overhead
- The model allows for slips and guesses
- The time complexity is linear
- Only a small number of parameters are involved.

(Shang et al. 2001).

**Designer interface**

The designer interface is the point of contact for those that design the knowledge base, the teaching strategies and store the presentation material needed for each lesson. If authoring tools have not been created to assist in this process then the material will have to be programmed in, probably by the creators of the tutor. A designer interface really implies that the tutor will be updated or used for a different purpose. Creating an authoring tool means coming to terms with the general concepts that make up the underlying components. Three noteworthy examples of authoring tools are Eon, PACT tools and REDEEM. Eon is noteworthy for its broad use, REDEEM because it fits neatly into the current mainstream teaching technology and the PACT tools because of their rule-based authoring.

**Eon**

At the most general level, Eon has a “Topic Ontology Palette” for defining a hierarchy of topics and sub-levels for a given ontology. There are different shapes available to represent different nodes and relationships can be labeled (“is_a”, “has_a”, “part_of”, etc…) Colours can also be defined for the links. Once defined,
the graphical symbols are available for use in an editor window. A hierarchical structure can be created by (or under the supervision of) the domain expert.

The “Interaction Editor” has an extendable palette of widgets for constructing the learner interface. The widgets have properties that allow for the wide use of graphics and storage for Contents objects.

A “topic/content” browser links the student interface components with their respective places in the knowledge hierarchy.

The student model has its own editor for establishing the values of certain student actions. These values are the basis for making pedagogic decisions. Rules about these actions can be written or altered in this window.

The teaching model is created using the “flowline editor”. Icons like “Decision”, “Animate”, “Erase/remove”, “Sound” and “Message Box” are available for constructing an event-based activity diagram. The diagram models all possible events for each atomic node of the knowledge being taught.

Tutors created with Eon include the Keigo “honorifics” tutor, the Statics tutor, the Chemistry workbench and the Refrigerator tutor. (Murray 1998).

**REDEEM**

REDEEM provides an interface designed for teachers. They can use the tutor after about 90 minutes of training. It expects that all the materials for the course to be
taught are available and created from an off-the-shelf commercial coursework product such as “ToolBook”. The authoring tools of REDEEM allow the teachers to describe the courses, construct the teaching strategies and determine which materials will be used for which categories of students (The student model is based on the Stereotypes technique.)

After the course, sections within the course and relationships have been described (the knowledge-domain), each section and “page” of the course material is described according to a set of criteria. Each criterion has a slider for setting a value (e.g. degree of difficulty). The ratings and relations are used for creating routes through the domain.

Figure 5. REDEEM’s Teaching Strategies Design Interface (Ainsworth et al. 1999).

An interface is available for setting default teaching strategies and defining new strategies. The default rules have been created by interviewing teachers. The teaching knowledge is ‘black box’ with the teachers able to overlay their own strategies. Easy exercises will be given early in the topic and harder ones later unless specifically overridden.

Excluding the creation of the course material, it took teachers less than two hours to create every hour of instruction, which suggests that the intention for REDEEM to enable efficient use teachers’ time was fulfilled. (Ainsworth et al. 1999).
PACT
The PACT Center Cognitive Tutor Authoring Tools for creating rule-based, model-tracing tutors require substantially more effort. A sound knowledge of functional programming techniques is necessary to encode the rules, but the resulting tutor can have as much knowledge as the user desires, with any level of granularity.

Below is an example of a rule that an author might need to create. The “buggy” rule defines a common addition error that a student might make - forgetting to carry:

```
(defrule buggy-add-addends
  (addition
   (problem ?problem))
  (problem
   (name ?problem)
   (subgoals $? ?subgoals $?))
?current-goal <- (process-column-goal
  (name ?subgoals)
  (carry ?carry)
  (first-addend ?num1 &:(neq ?num1 nil))
  (second-addend ?num2 &:(neq ?num2 nil))
  (column ?column)
  (sum nil))
=>
  (bind ?sum (- ?num1 ?num2))
  (modify ?current-goal
   (sum ?sum))
  (printout t "Add addends." crlf)
)
```

Figure 6: The CLIPS-like functional language used to create rules in JESS, the Java rule engine. (Choksey 2003).

The designer interface is a combination of seven frames, some with different modes, combinations of different menus and button actions, acronyms and application-specific jargon. Some of the windows must be used as a group. While the interface can be learnt with intensive expert attention, the online tutorials are less effective and reflect the complexity of mastering the interface. Developing rules that describe the knowledge to be learnt is the real issue. A seemingly simple task like telling someone that they have forgotten to carry requires much expertise.
Student interface

Much research has gone into HCI and nowhere are the concepts more important than in the process of learning. Issues such as ease of use, cognitive overload, use of screen real estate, consistent appearance, useful and “intuitive” tools, and optimized help must be considered in almost every GUI design. Specific designs extend learning theories for specific applications. Ego, for example, allows the student to backtrack if they make an error. They can discover the answer themselves instead of being told. (Ng et al. 1995). Collaborative learning can be carried out with chat interfaces, problem solving can be enhanced with statistical graphing tools and procedures learnt through the manipulation of virtual reality simulations. Reading tutors can listen. (Features of these systems have been mentioned above.)

Most interfaces include a place for feedback to be viewed by the student, opportunities to seek assistance, often a view of their performance to date and a map of where they are situated in the course. Material that will need to be entered exactly for evaluation or storage is provided as option widgets (e.g., combo boxes, menus, sliders, check boxes and radio buttons). The virtual assistant is very common.
Virtual Assistant

Virtual assistants are frequently used to deliver feedback to the learner in a way that makes the interface friendlier by creating a more human-like interaction.

In the cockpit-like interface for simulating engineering exercises in mechatronics, a virtual assistant is employed as a tireless go-between, providing feedback as well as communication between students, tutors and lecturers. (Geoffroy et al. 2002).

Don Pancho or Dona Rita are both agents that represent basic emotions such as happiness, sadness, anger, surprise and disgust. Changes in emotion accompany the primary school students’ interaction with the Tutor GUIs. The agents were found to have a positive effect in guiding the students to particular concepts without forcing them. It was hypothesized that the role could have been successfully expanded so that the assistants also offer “interesting paths and external learning resources.” (Zapata-Rivera and Greer 2002).

In SIMPA the work-station trainer the professor and his companion parallel the instructor and the student and are linked to the student’s performance. They hold a dialogue as a result of questions asked by the companion. The companion asks those questions that the student should have asked, but didn’t. This draws issues to the attention of the student or raises issues that the student is hesitant to mention. (Duquesnoy et al. 2002).

Delivery of Software

The context in which an intelligent tutoring system is to be used will have a bearing on the architecture chosen to deliver the service. While the following considerations for determining an appropriate architecture are listed separately, they will obviously be interrelated. As most ITS are initially research tools, delivery will be strongly influenced by the context and constraints of the experiments for which the tutors are designed. Delivery mechanisms may also vary according to:

- Type of subject matter and software architecture
- Type of student
- Type of hardware/network facility
- Type of education provider
If a high-speed link with sufficient processing power to meet demand is available (and high demand flows are given priority) the distinction between a stand-alone application and a distributed one becomes transparent. The advantage of running an entire stand-alone application on a local machine disappears.

There are significant advantages to using a distributed architecture from the point of view of the service provider. Centralization of key components allows for the reuse of data, eliminates the requirement of having to update multiple copies when adding modules, and allows for the sharing of computing resources and data.

An interesting factor is that the quality of functioning of a tutor may be improved by collecting a large, varied set of sample cases from multiple sources. If it has a case-based machine learning module included in its operation, an ITS will better refine its capabilities by collecting cases in a central repository.

ASIMIL, a flight simulator collects error cases that have already been processed and reuses them if the same error arises again, saving on processing and latency of reply. (Aka and Frasson 2002).

The distributed models used in Intelligent Tutors build on either traditional Web-Based technology or new technologies that allow applications to interact in a more autonomous manner. Agents can behave independently and collaboratively. They can also be mobile on a network. The agent can be seen as situated in its own context, interacting with other agents as part of a community to solve a given problem. Intelligence is often viewed as “emergent” – the result of problem solving by a community of agents which is greater than the sum of its parts. (Luger and Stubblefield 1997).

**Traditional Web-Based, 3-Tier Architecture**

The World Wide Web, given its ubiquitous nature, is already involved in the remote distribution of Intelligent Tutoring Systems. It provides a means of teaching that is platform, location and time independent. There is now a vast infrastructure available to educators and students.
The number of students that can be taught at one time is only limited by network capacity and computational power.

The traditional delivery architecture for the World Wide Web has been to run an HTTP server with a database. Upon request, the server runs a script that accesses the database for appropriate data and returns a dynamically generated web page to the client that made the request. The server starts multiple processes (on different ports) to handle concurrent requests. Java applets, Flash files, etc can be embedded in a web page to act as an interface. Applets and JavaScript can be used to run some of the programming needed for the tutor to operate on the client machine. Technologies used for the server-side scripting that makes these ITS adaptive include ASP, PHP, JSP, CGI and Servlets.

**Agent distributed**

Once the notion of modules and intelligence is blended with the distributed system it is not a great leap to consider ITS components as agents themselves.

Many of the intelligent tutors claim to be using “Agents” when in fact they may be cousins of agents – parts of a concurrent/distributed system. The distinction drawn by Wooldridge is that agents are autonomous and make their decisions at run time whereas the synchronization and co-ordination structures in a regular distributed system are hardwired. Secondly, Agents encounter each other as “self-interested entities” rather than being assumed to have a common interest in making the overall system function correctly. (Wooldridge 2002)

Regardless of their title, some ITSs are built as distributed systems based on either of two styles of technology: tuple-based or RPC–based. Tuple-based systems use a central space, like a database that various agents write and read to. This allows them to exchange parameters according to either a closed set of rules, or via an open “standardized” means of exchange. JavaSpaces is one technology that implements this approach. WCL, KQML and other domain-specific languages are used for communication. (Papadopoulos 2001)
RPC (the original Remote Procedure Call for distributed systems) has been adapted to deal with Object Oriented concepts of message passing to invoke and receive services from a remote server. This model is not all that different from a normal HTTP web server model, with the exception that it uses object and message exchange protocols rather than a text-based exchange. DCOM, .NET, CORBA and RMI are technologies that can be used as middleware brokers for “agent” exchange. (Oberg 2001). While HTTP can be avoided, these technologies rely on the TCP/UDP transport layer of the IP suite of protocols to shift their datagrams around the Internet. It is possible to integrate an HTTP-based component to such a system.

As mentioned previously, FIPA has provided widely accepted standards for agents and JADE implements these standards in Java using RMI middleware.

**Stand-Alone Applications**

The main value in using a stand-alone application is when the processing and network services cannot support a distributed system. Any intelligent system involved in parsing a semantic or hierarchical network of nodes is going to require a deal of processing ability if it is to remain viable when expanded fully. If multimedia and communications play a major role in the learner’s interface, then having a dial-up 56kbps connection will be a likely bottleneck. In these circumstances a stand-alone application may be a better option.

Users of the reading tutor LISTEN spend 20% of their time waiting for the tutor to process the input and respond with the required feedback. (Mostow et al. 2002). A situation like this where auditory data has to be parsed and evaluated over a 56kbps modem would be completely unworkable. (The evaluation of ITS performance below deals with response latency in more detail.)

The major drawback, of course is that a stand-alone application cannot be readily modified. In this situation, Java’s Web Start technology might be useful. If the program is not outrageously large – an uncongested modem connection will download 12-18Mb in an hour – it is conceivable that the application be downloaded from a central repository. Web Start always updates to the most recent version and no installation procedures are required. (Sun Microsystems 2003). Newmarch has
used Web Start to download working Java programs for students to manipulate locally. The entire environment needed to run the program is established and sandbox security is created on the local machine. The disadvantage of the system is its inability to permit socket creation and thus networking from the downloaded program. (Newmarch 2002).

**Review implications**
While there has been a deal of research on how to model students, knowledge and teaching with a view to making effective intelligent tutors, there has been little mention of just how these tutors perform in terms of processing time and bandwidth. The latency between user input and tutor response will be a major factor in determining the educational value of a given tutor.

If deployment of one or more ITS is to be considered, then the type of delivery method used may depend upon what performance a Tutor can deliver. Similarly the architecture of the tutor one chooses when implementing a tutor may depend upon the capacity of the available hardware. If delivery must be by dialup connection, or the server has little processing power available, then performance issues are critical. A review of the ITS literature was unable to inform any decisions about deployment based on computational viability.
Performance: Survey and Studies

ITSs are mostly research tools that have not been subject to public testing in the same way as a commercial product. In order to assess the viability of deploying an ITS an evaluation of the computational performance of these programs is necessary. An initial survey of existing Intelligent Tutoring Systems was completed, focusing on likely performance issues.

Questionnaire design and administration

Since academics from many nations would be asked to participate at a variety of points in the academic year and that communication was from an unknown source via email, it was important to make the survey short and easy to complete. Questions were designed to be electronically dispatched, filled in by voluntary participants and returned by email. Potential participants were given a choice of formats when asked if they wished to volunteer. Hard copies of the survey were also included in the registration packs for the AIED 2003 conference in Sydney. Below is the MSAccess form, with dropdown selection lists.

Figure 8: MS Access version of ITS Performance questionnaire.
Questions contained clarification internally when considered necessary. In the case of potentially open ended questions, a choice of likely answers was given and an “other” option. An opportunity to qualify answers was deemed appropriate, given the diverse nature of the systems, and space was provided on the form.

The questionnaire could be regarded as a pilot survey to determine whether further research was warranted and if so, to provide an opportunity to identify pitfalls and refine questions.

**design of questions**

It was assumed that there would be significant detectable performance differences between different types of ITS. Variables that were likely to affect performance were identified and questions about these were included in the questionnaire. It was hoped that answers to these questions might produce clusters to help identify the different “types” of ITS and their attributes. In addition to these variables, three educational features were seen to be important in the deployment of a tutor. These features were the type of feedback given to the student, the distribution method of the ITS and whether an authoring tool should be used. If there was any association between these factors and performance, it should be detected by the questions.

**Indicator of Performance**

“Response latency” was chosen as the most suitable indicator of performance because it represented the critical (and frequent) interaction between the user and program as well as requiring the involvement of the key modules in the system. Response latency was defined in the questionnaire as:

“...the period between when a student submits input for evaluation and when the response is received...”

Response Latency is a measurable aspect of performance, but evaluating the importance of a given performance level is not so easy. Fine differences in response times may not be important differences. Ultimately adequate performance is judged by the user. Intuitively, if the student finds that the system is performing so slowly that it is frustrating one would expect this to impact on the quality of the learner’s
experience. A corollary of the active learning/constructivist theory of learning, (that active involvement leads to construction of meaning) is that inactivity results in diminished involvement and therefore diminished learning. A student engages in active learning when using an ITS by engaging in the exercises presented by the tutor, focusing on the content, engaging in self dialogue (including metacognition) and engaging in dialogue with the tutor. If delays in responding to the student are great enough to be distracting, then self dialogue as well as task activity will most likely diminish.

A user’s response will depend on many psychological aspects including his/her expectation of the system. Regular indications of a task being completed, such as a progress animation, can temper users’ expectations of instant service. (Klein et al. 2002).

There have been no studies that attempt to untangle the complex relationships between variables such as frustration, expectation and attentiveness with respect to learning and response latency. Ass. Prof. Dr Nancy Alvarado, whose interests lie in this area, pointed out that if time is of no concern to the user then slow programs are not a problem. She also suggested frustration may be sufficiently but not necessarily connected to aggression. Aggression and anger may or may not be connected to cognition. The question “What is satisfactory performance?” remains open. (Personal communications, 4-16th September, 2003).

There are a few metrics in the literature suggesting what appropriate times might be.

“Feedback should be commensurate with action
   –Echoing input < 0.1 secs
   –Page turning < 0.5 secs
   –String search < 4 secs
   –Provide time affordances where delays are inevitable
   •Where delays are inevitable and unpredictable, users need reassurance that an operation will complete
   •User sensitivity to delay depends on context
   •User interface should provide time affordances such that delays become part of the accountable behaviour of the system”

(Procter 2002).
The current Human Computer Interface wisdom according to work by King (quoting Schneiderman and Shackel) suggests that there are three levels of satisfactory web-based performance, depending upon the task being performed and the user’s expectation that this is indeed what is being performed. The first is instantaneous, for tasks such as typing or engaging in dialogue where speed of communication is basically as fast as the user can act. The next is the 1 second level, where a change or move is expected. The slowest is about 10 seconds where the user expects the software to do serious searching or processing of some kind. (King 2003).

Most tutoring that requires in-progress immediate feedback emulates a human tutor and might be described as thoughtful dialogue. Ideally this would occur in less than a second. Evaluating an entire exercise is not so frequent, being only at the end. The user will accept a delay in the order of the slowest level without any major impact.

A response latency of up to two seconds was considered by Casual Lecturer in HCI, Kevin. Manderson, from personal observations to be a satisfactory web-based delay with delays beyond that likely to impair the educational value of an activity as the user would be increasingly disengaged from the content. Anything beyond seven seconds he considered to be completely unworkable (personal communication, 25th May, 2003).

It is likely that regular use of Internet modem connections (that travel at the speed of a 56kbps) may set up in the user a degree of tolerance for slower screen changes. Again no concrete evidence could be found for this hypothesis. For the purposes of an initial study it was considered that one second or less would be ideal and that slightly more would still be satisfactory. Further, the responses to the survey were likely to be estimates and thus unlikely to have the degree of accuracy to discriminate between ideal and satisfactory. Participants were therefore given the following choices:

0-2 secs; >2 – 4 secs; >4-6 secs; >6 secs
Variables

A number of variables were identified as likely to have an impact on the response latency of any given ITS. These were:

- the number of decisions that the system had to make to provide feedback
- whether heuristics were used to reduce the problem space
- the time complexity of the processing algorithms used (the ‘Big O’)
- the number of processing modules involved in the system
- the speed of the hardware on which the ITS was evaluated as indicated by processor speed
- whether the system is written in a compiled or interpreted language
- The size of new pages – especially for networked applications.

The decisions made by the system can be expressed in many ways such as rules, graphs, and frames. Decisions may be based on rules generated by the system as is the case with Bayesian probabilities or fuzzy sets. The multitude of different methods for getting the ITS to adapt to user behaviour and learning are embodied in the way the decisions are made. The number of these decisions might be important to understanding performance during evaluation. Participants were asked to estimate the total number of rules used by their system.

It is standard practice for systems with large sets of decisions to ensure that they process a minimum number of total choices in the problem space. A number of heuristic approaches are used to this end. Four were listed – “Expert/Experiential”, “Neural”, “Bayesian” and “Hybrid”. The participants were given the option of “other” and the opportunity to elaborate.

Computational complexity is a major variable in determining the performance of a system. The order of complexity relative to the number of decisions made by the ITS was anticipated to be a suitable indicator. A large number of options ranging from constant time to exponential time were given and an “other” option. The degree to which the system is an experimental one as compared to a mature piece of software might be seen in the optimum choice of computational algorithms. Participants were
asked to rank the maturity of their system on a scale from 1-4 based on the degree of 
optimisation left to do.

ITSs vary considerably in their structure. Most systems developed recently are 
modular in nature, with different functions ascribed to different modules. These 
modules exchange messages. The number of modules might be a predictor of 
performance. Participants were asked the enter a number “functional modules” in 
their system.

Processor speed was chosen as the indicator of speed of the hardware. It was 
assumed that the bus speeds, memory types and quantity, etc were those of a 
properly functioning high quality machine built to integrate the processor of its day.

Compiled programs run faster than those applications that require an interpreter. 
Participants were given a choice of these two options.

If a 56kbps modem is considered as likely networking hardware, then the size of 
download and upload of information becomes a performance issue. Survey 
participants were asked to estimate the size in kilobytes of a typical content page.

**Additional Considerations**

Three additional features were considered to be central to any decision to deploy an 
ITS. The first was the nature of the adaptiveness of the system to the user. The 
educational value of an ITS, as opposed to any other form of computer based 
instruction, is the way it personalizes its interaction with the student. Student 
modeling and feedback were considered.

Evidence provided in the review of educational literature pertaining to ITSs, 
indicates that it is important for some form of student modeling to be open to perusal 
by the student. (Hartley and Mitrovic 2002, Zapata-Rivera and Greer 2002). The 
student is then able to think about the way in which the computer perceives his/her 
learning and evaluate it. An interface that promotes (almost insists on) metacognition 
provides additional educational gain. The study checked to see if there was any
likelihood of a correlation between performance and the presence of an open student model. Participants were asked if their system included an open student model.

The way in which feedback was delivered to the student was also considered worthy of scrutiny. Studies on the merits of immediate and final feedback mentioned earlier indicate that long-term benefits might be greater for final feedback. (Mathan and Koedinger 2002, Mitrovic 1998) It would be unfortunate if the choice of feedback methods had an effect on the performance of the systems and limited deployment options. Participants were asked to identify the type of feedback used by their tutor and given a choice of “Immediate”, “Final”, “Immediate and Final”, “On Request”, and “Inherent”.

The second feature considered important when deploying an ITS is the choice of distribution methods. If a form of distribution significantly limited performance it would have deployment implications. There are many potential ways to distribute tutors in particular many hybrid forms of networked architecture. Problems of categorizing distribution such as “Is a tutor that is delivered via Web Start standalone or distributed?” were left to the discretion of the participants. They were given four options and asked to choose the one that best described their system – “Standalone”, “Traditional 3-Tier Web”, “Distributed Agents”, “XML Web Service”. These broad classifications represented both mature and developing distribution technologies. Again, the option to qualify choices was available.

Finally, authoring tools were considered worth exploring as there are distinct advantages in having a system designed for two types of users rather than have the teacher start anew each time an ITS is deemed to be useful. Although seemingly unrelated to performance, it was considered worth ensuring that no association existed. Participants were asked if they had used an authoring tool to create their ITS.

**Observations**

Only twenty-one surveys were completed so no conclusions could be drawn statistically. Wide variation in the sample systems made any sort of generalisation difficult. For example one tutor, Teatrix, was a collaborative ‘theatrical agent’ and had no regular response time. A number of the tutors such as Excel tutor were
primarily designed for cognitive research experiments rather than specifically as robust tutoring software. The comments provided by the authors highlight the differences found and have been included in Appendix A as well as the main groupings used below (Appendix B). An example is the Disciple system with

“586 objects and features, 409 tasks and 419 complex task reduction and solution composition rules.”

Evaluating clusters of results is at best only a basis for further study so groups of three or less items were totally ignored and the comments below are only observations that might be tentatively made from the larger clusters of instances.

The key question of response latency indicated that most of the ITS returned within acceptable measures of response. Of the 20 systems, 14 ran at the highest level (0-2 secs), 17 in the top two levels (0-2secs, >2-4secs). The assumption that there would be significant detectable performance differences between different types of ITS does not hold in the light of these results.

**Resultant clusters**

The types of distribution systems suggested that Stand-alone and traditional 3-tier Web-based systems were in roughly equal proportions and accounted for 17 of the 20 responses. They were spread in a similar fashion across the spectrum of performance. The small number (3) of distributed agent systems were all in the top performance range, but the number is too small to draw any inferences. No responses indicated distribution of intelligent tutoring as an XML Web Service.

All four of the ITS built with authoring tools were reported to perform at the top level.

Of the thirteen ITS authors that knew the speed of their systems, the average speed (ignoring one outlying result) was 580 MHz – about _ of the speed of current machines. Performances on today’s machines will most likely be better than those reported. It is reasonable to believe that most optimized tutors will perform a one second “thoughtful dialogue” on the latest hardware.
The concept of “content page size” didn’t always map cleanly to the surveyed system. Jack Mostow’s LISTEN reading tutor in particular returns sounds that have no typical size. Many of the systems exchanged only a few lines of text, not even amounting to a kilobyte. Ms Lindquist spreads its data over two frames – one for maintaining the student model and the other specific problem feedback. A large group (12/20) fell within the 0 to 2KB range. The remainder fell into the 18-65KB range (5/20) or the very large size of near 0.5MB (3/20). Of the tutors with large content size two are applets (WADE-IN and Logic-ITA), downloaded once, and the other (Excel-Tutor) is a research tool that operates as a stand-alone executable.

Fifteen of the systems either had no or only minor optimisation of code still to go. Eliot pointed out that he did not deliberately set out to optimise the code of his Stochiometry tutor, but automatically used efficient algorithms during implementation. This might help explain why fourteen respondents were not aware of the “Big O” of their main processing modules. Given considered implementation and adequate performance, working out the order of performance is unnecessary.

The compiled versus the interpreted languages performed similarly. Many of the ITSs are Java based. While it is reasonably well documented that Java, using JIT HotSpot, is around three times slower than a C++ executable, this seems to be unimportant with regard to the satisfactory performance of an ITS.

Notably the presence or absence of an open student model did not correlate with any change in performance. The type of feedback was not associated with a given performance, either. It would seem that these features can be used as deemed educationally appropriate without significantly affecting the performance of the system.

Similarly the decision about which distribution method to use can be based upon the context in which the tutor will be deployed. The decision need not be constrained by performance issues.
The issue of network constraint upon the performance of these tutors is really only relevant in the case of delivery via 56kbps modem. The tutors likely to perform slowly when connecting via modem delivered a large initial payload and then exchanged text.

Conclusions about survey questions and methodology

Clearly there is scope for further exploring the range of tolerances to response latency. A design of two variables, learning measured vs. time delay, is conceivable. Trying to untangle frustration, anger, aggression, time delay and amount learnt would not be as easy. Given the results of the initial survey this matter is hardly worth pursuing since most ITS performances are within the limits of currently accepted response times. It is questionable whether any person who is frustrated by a response time of 1-2 seconds is actively learning something. In which case, one must question the value of using an ITS in the first place.

The only key omission that needs to be addressed is that of load on the server’s resources, raised in Brent Martin’s comments. The SQL Tutor builds the solution to the problem on the server in order to evaluate the student’s solution. How many sets of decisions (concurrent users) can the server support? Key variables include the size of the decision graph, number of frames/objects or size of the symbolic rule set and the algorithms used to manipulate these. Aka and Frasson’s flight simulator ASIMIL uses a clever technique for minimising the size of the network communications. The simulation on the client machine also runs on the server with only text passing between the two. (Aka and Frasson 2002) A number of concurrent requests to the server might conceivably stress this system too.

One possible approach to the problem would be, given the resources, to pass overload to another server. Another solution is to prevent the problem from occurring in the first place by distributing the systems to the client machine, by deploying ITSs using a decentralised model.

Additional Performance Components

Additional performance data was gathered to corroborate the findings in the survey.
Performance figures exist for instances of external components that might be used as part of an ITS. In particular a database for storing user information, a user modeling server, has recently been tested under rigorous real world conditions. The production rule engine, JESS, has also been benchmarked. Data is also available for inter-agent messaging using the JADE technology. Comparative studies of different Internet-based client-server technologies also exist. These details follow.

Kobsa and Fink (2003) evaluated their user modeling server in the light of an analysis of user web browsing behaviour. They emulated four request frequencies for 100, 500, 2500 and 12500 user profiles. Their server included components to provide directory services, analyze user behaviour, discover user interests and draw inferences about the user based on rules. The components approximate the types of processes one might expect to see in an ITS with the exception that the knowledge domain upon which the rules are based might be more extensive in an ITS. The server was able to customise a page in a mean time of 52.57 milliseconds if asked for 4 pages a second and having to search a database of 12,500 profiles. A school or university student modeling server seems unlikely to add much burden to ITS performance and may be a way of enhancing it. (Kobsa and Fink 2003).

Ms Lindquist and a number of other tutors generated from the PACT authoring tools (by the Carnegie-Mellon researchers) rely for their intelligence on a rule engine called JESS. The rule engine is used to trace a student’s progress through the knowledge domain and determine the feedback to be relayed to the student interface. Performance data for JESS is presented by Friedman-Hill

“Using Sun’s HotSpot JVM on an 800 MHz Pentium III, Jess can fire more than 80,000 rules per second; it can perform almost 600,000 pattern-matching operations per second; it can add more than 100,000 facts to working memory per second; and a simple counting loop can do 400,000 iterations per second. Independent benchmarks have shown that Jess is significantly faster than many rule engines written in the “faster” C language. For example, on many problems, Jess outperforms CLIPS by a factor of 20 or more on the same hardware.” (Friedman-Hill 2003).
The largest number of rules - or decisions - that any of the tutors surveyed used was 800. Based on the above data these would be processed in 0.01 of a second.

If there are to be any major performance issues they are unlikely to be with the background processing issues. They are more likely to be with the delivery process.

Tests done on two 800Mhz PC’s connected by a 100Mbps VLAN were carried out by Cortese and colleagues at the TILAB in Naples in order that benchmarks be established for JADE inter-agent communication. Results included times for an agent on one platform, running in its own JVM to send a message and receive a reply from a similar agent on another machine. The JIT Hotspot compiler was included in the tests. Different FIPA-compliant messaging protocols were used with marginal difference and a comparison was also done with RMI messaging. The latter test indicated the difference between the standards overhead and raw Java performance. One thousand messages took about 7 seconds (7 milliseconds per message) using JADE agents. RMI managed to complete the task in a little over 4 seconds. Arguably this experiment occurred over a very short distance without satellites, denials of service and so on, but it did demonstrate that there is not a lot of cost attached to the use of agents for messaging purposes, once the message has been composed. (Cortese et al. 2002).

Boian compared various Java Internet technologies with a compiled ANSI C CGI program. The experiment recorded the time taken to request a counter action on a remote machine and get the response. CGI, Servlet, and JavaSpace were compared. Tests conducted continuously over 60 minutes showed the average response times in milliseconds to be CGI 15, Servlet 14, and JavaSpace 54. CGI completed 234,000 transactions, while the Servlet completed 260,000 and the JavaSpace application managed 67,000. (Boian 2001). Even if a distributed application uses the JavaSpaces option for sharing centralised resources, the processing cost will only involve one twentieth of a second.

Studies of technologies that have been used to implement components of ITSs can be seen to confirm the estimates of performance of complete ITSs made by the authors surveyed.
Proposed Architecture

This thesis has shown that deploying an ITS, even belonging to the domain of research alone, is viable both educationally and computationally. The major potential restriction on performance that emerged was insufficient server memory for multiple requests. A number of solutions to server limitations exist. The distribution architecture presented next will provide one solution.

Context of Architecture

As there appear to be no technical obstacles to the success of an ITS deployment that cannot be worked around, the next practical step is to consider the context in which the ITS will be deployed.

Centralised Institutions

Currently the use of fully centralised and co-ordinated institutional deployment is rare but being developed by the larger research institutions. A good example of a centralised approach to deployment is Knowledge Tree, where a fully adaptive web portal integrates WADE-IN. The tutor is integrated with a centralised student modeling server and a series of activity servers. They deliver tutor services as well as disseminate general course materials related to aspects of computer science. When a student selects a topic from the portal, the selection as well as the centralised student model is used to choose and deliver the material best suited for the student. (Brusilovsky and Nijhavan 2002).

The University of Massachusetts has two centres for the development of ITSs. According to Assoc. Professor Beverly Park Woolf, they produce tutors for individual schools, but have no centralised facility studying student performance and integrating the tutors into a unified offering (personal communications, 4th September, 2003). Similarly Sydney University has a number of tutors in the School of Information Technologies and a user modeling server, but have not (as yet) integrated them, according to lecturer, Dr. Kalina Yacef (personal communication 6th August 2003). There are a number of advantages to being able to data-mine the student model including analysis of class performance on topics (when classes are
large), automating assessment, and the ability to selectively retrieve appropriate documents based on the ‘types’ of students who respond favourably to them. (Tang and McCalla 2003).

Figure 9: Description of the broad functions of Knowledge Tree. (Brusilovsky and Nijhavan 2002).

There is no reason why the huge amount of development necessary to produce a successful ITS causes an ITS to be tied to an individual institution – even if it is intertwined with central user modeling. Current research in education is moving away from the ‘instructor centric’ approach to learning and moving to the ‘learner centric’ approach. It is possible to allow the user model of a student to be stored on the student’s local machine and any details that need to be used centrally, to be duplicated. (As mentioned, one important use of the centralised repository is to store user data for case-based reasoning modules that might be integrated with the tutor.)
Recent developments in standardised agent technologies and learning object standards are available to facilitate the freer (not necessarily ‘free’) approach to the distribution of Intelligent Tutoring Systems.

**Description: Components and Processes**

**Web Service Yellow Pages**

The context of the deployment of an ITS is such that it will most likely be accessed not just via an individual institution’s portal as part of a curriculum, or an agreement between institutions or be discovered by a search engine, but it will be able to be found in an XML Web Services (‘yellow pages’) index by an agent acting on behalf of either an individual student or an institution. In this architecture, each ITS entry would hold standard IEEE Learner Object Metadata (LOM) details like contact, name of Tutor, author, size of software, distribution method, system requirements to run it, description of the knowledge domain that it services and the cost of the service. The IMSGlobal DTD can be used for implementing this standard as an XML binding. The yellow pages will have a URI, a copy of which is held by a Broker Agent. Each catalogue will be relevant to a general learning area and indexed by the general (higher level) concepts of a given ontology. (IMSGlobal 2000).

**Broker Agent**

The broker agent will have a list of many educational yellow pages directories and will visit the most appropriate of these to find a list of ITSs. The tutor might be a web page (form-based, server-side scripted) application, an applet on a website, a downloadable stand-alone application. The tutor could also be another agent or collective of agents in a container at a given URI.

The broker agent, acting on behalf of the institution or student, would match the knowledge domain desired and the system requirements available with those found in the yellow pages and return a list of available options – prioritised according to the quality of the match. This is analogous to a simple customer and vendor relationship, only the product is an educational utility and may be subject to more complex contract/licensing arrangements than a single software product. The LOM
model allows for this sort of information to be included in the tutor’s yellow pages listing.

**User Agent**

The end user, given the prioritised list of hyperlinked available tutors, would select one and be led along the distribution path required by that tutor. The ITS would complete whatever communications were necessary with other locations. In a decentralised scenario this would imply reading and writing to a student-model file on the student’s computer. The user agent then must have a means of communicating with the broker, an interface for the display of results and two-way access to the student model files stored on the user’s machine.

It is worth noting that Agent activity is not confined to desktop computers, but is also possible on mobile devices. The 3G mobile devices will have Internet capacity beyond standard modem speeds. (Qualcomm 2001).

**Web Start and JADE**

The environment chosen for prototyping the agent interaction described in this architecture is that provided by the JADE project. This environment implements, in Java, the standards for agent behaviour and deployment specified by FIPA. Java was also preferred because of its cross-platform capability, links with XML technologies, public domain presence and fast compiler.

The agent-based distribution also provides an ITS prototype that can be downloaded to the user’s desktop using the Java Web Start technology via 56k modem. The advantage to this approach to distribution is that the tutor, although having to be reproduced in persistent form, does not have to be reproduced in volatile memory, thus reducing server load and any potential performance problems. Web Start overcomes the disadvantage of copies becoming obsolete by providing the user with updated versions of the software. As mentioned earlier, the drawback is that the tutor must be designed to operate without socket facilities. If necessary, other methods of communication will have to be implemented if centralised communication is necessary.
Figure 10: Model of component interactions for agent-based deployment architecture.

**Example Tutor**

The description of an example Intelligent Tutor follows. This tutor will be able to be downloaded via Web Start and perform its functions locally, avoiding the need to communicate with a centralised server. The ideas are a skeleton upon which many refinements are possible. Primitive prototyping is all that was used for purposes of implementing the deployment architecture.

**Role**

The example ITS planned is a computing algorithm tutor that allows a student to practise the steps needed to build the algorithm without writing the code. The student will have met the algorithms either in readings or as part of a lecture and is given the opportunity by the tutor to consolidate the knowledge. The tutor is flexible enough to accept modules for any level of algorithm and thus is useful at any stage in an undergraduate or upper secondary computer science course.

**Educational approach**

Each algorithm is a discrete module including one or more exercises, the solutions of which implement the algorithm. The ITS provides the student with a list of algorithm “steps” from which to choose (e.g.: “Initialise the stack structure”). The list includes distracters, requiring the student to be actively engaged in the construction process.
Construction is reinforced by presenting an implementation in code when the student correctly selects the next step of the algorithm.

The degree of generality of feedback and the algorithm steps themselves can be adapted to the errors made and the level of knowledge displayed by the student. This will enable the student to operate within their zone of proximal development. Multiple exercises with varying degrees of difficulty can be introduced by the instructor. Feedback is available immediately and on request. The student will be able to view the system’s model of his or her knowledge. Words or symbols can be used when tailoring exercises to suit the concepts to be taught.

**Interface**

The user interface design has two panels beside each other in a single window. The leftmost panel displays the selected problem at the top and provides a combo box with a drop-down list of algorithm steps from which the user can choose below the problem. If a step is selected correctly, that step is displayed in the rightmost panel as a code comment, with the code for implementing that step immediately underneath. If a step is selected incorrectly, feedback is displayed in the feedback textfield of the leftmost panel, below the combo box.

The menus for the interface are intended to allow the user to open and close exercises, print the code generated in the rightmost panel, view the student model kept by the system, and import new exercise modules. Help and software information can also be accessed from the menus.
Knowledge Domain

The level of granularity of the knowledge domain within the tutor begins with the concepts behind “steps” that a student is required to select. All algorithms have a precondition a process and a postcondition concept. For a binary search, as an example, concepts might include:

- **Precondition**: Sorted array.
- **Process**: Concept_A reduce search Space by half, Concept_B array midpoint and check, Concept_C recursion.
- **Postcondition**: Return Boolean.

Finer levels of granularity can be introduced if necessary.

Once these concepts have been established, the pedagogic aspect of the tutor can be built and the evaluation of student knowledge becomes possible.

A description of the knowledge domain could be created as part of an XML file that also holds the pedagogic content (or the URI’s of content).
Pedagogy
It is assumed that the finer the detail to which the student is exposed the more understanding is expected of the student. A better student will have a refined understanding whereas a less knowledgeable student is most likely to suffer cognitive overload if the concepts are not at the more general level.

“Steps” are the items that the student must select and these can vary in the degree of their generality. A set of steps has a level of difficulty. There will be fewer of the least difficult steps needed to complete an algorithm than the more specific steps. If a student is not suited to the step level it can be changed in mid exercise. If the student finds the work too hard, then the tutor recognises errors, and if feedback is not effective, reverts to the first incorrect concept and starts the steps at a more general level. If the concept is grasped without error, then the next concept is presented at a more difficult level. Exercises repeat the same concepts in different scenarios. (There is no need to graduate the difficulty of the exercises as well.)

To extend the example of binary search, the following steps for process/concept C might be given:

- Easier (more general) “Find the midpoint of the array and see if it is the target”. This should be selected by the student from a set of distracters such as “Check that the first term is not the target”.
- Harder (more specific) “Get the index of the first term and add it to half the size of the array.” and a similar distracter might be “Subtract the index of the first term from half the size of the array.”

The feedback offered by the system is based on the principle that the most effective learning is done when the student is actively involved in thinking about the problem. In order to achieve this it is customary to provide general assistance first and, if necessary, more specific clues. So called “buggy rules” can be used to manage predictable misunderstandings.

Binary search requires a search of either the lower half of the array or the upper if the target of the search is not the midpoint. If the size of the new (half) arrays was incorrectly selected as “original size divided by 2” then the following feedback could be given: “The midpoint should be in one or the other of the two new arrays.”
student is left to ponder the implications of the feedback and perhaps revise the algorithm.

Each individual algorithm exercise set would be stored as an XML file, with the concepts, buggy rules, steps, levels and feedback encoded. Subsequent development of new algorithm tutorials could be created using the document type definition (DTD) or schema in a standard XML editor. XML permits the distribution of the exercise separate from the tutor, so a user can download new exercises as they become available and open them with the Algorithm Tutor. The system would also be able to be rebadged as a “Binary Search Tutor”, a “Shortest Path First Tutor”, etc when fully developed. A portion of a DTD has been included as Appendix F to illustrate how this can be done.

**Student Model**

The level of achievement of the student is recorded as the name of the concept being learnt and the level of the current set of steps being used. Errors anticipated by buggy rules are also included. The student model is a subset of the pedagogic one and as such can be expressed easily in an XML document. The model is not connected to any other exercise or centralised assessment system, although the data is stored in a means that is transferable and readily understood by those responsible for generating the exercises.
Distribution Implementation Outcomes
A simple proof-of-concept approach was taken to this implementation. The edges of the implementation, the student requirements and the Yellow Pages, were set as files. Separate JADE containers to hold each agent were created on a single management platform.

The Yellow Pages was created as a simple delimited list of URI’s in text format. The student requirements were implemented using the IMSGlobal LOM definition for learning objects. (The LOM definition includes scope to define an ontology in which to locate both the educational value of the tutor and the value that the student is seeking.) The Agents operated in a confined environment defined by the JADE application and communicated using Java serialization. This environment can be readily expanded to multiple platforms and the communication method can be defined in XML to cross environments and agent systems, but this degree of complexity was considered unnecessary.

The Agent Context
Three containers were used to hold the agents: the management container, the Student_Agent container and that holding the Broker_Agent. The management container incorporated the JADE graphical user interface and allowed visualisation of the process and the creation of sniffer agents to detect and record the messages passed between the agents. In the screenshot below (fig 12, the Student_Agent checked with the management system for the address of the Broker_Agent. The management system responded. The Student_Agent passed a request to the Broker_Agent which replied.

Agents are described in terms of state changes. Each change of state is a new behaviour. Typical behaviours are defined for certain types of agents. The Student_Agent was implemented as a typical client agent and the Broker_Agent was created following a server template. They were not given mobility. The state change diagrams are included as Appendix C. The key aspects of this implementation
extended the “request” and “receive” behaviours of the Student_Agent and the “process-request” behaviour of the Broker_Agent.

![Figure 12: showing the separate containers and agents in the left window and the messages passed between the agents in the right window](image)

The Student Model

The descriptor of the user’s request was created using the IMSGlobal XML implementation of the IEEE’s Learning Object Metadata standard. This allowed the setting of the user’s requirements to include keywords as well as LOM defined terms such as “University First Cycle” and “no” (cost).

In addition to the user’s requirements XML file, the model object contained an integer used to identify the request. While this was not crucial to the exercise it represented the possibility of expanding the model to include aspects other than just the description. For example, it might be possible to add a maximum price that the user is prepared to pay, contact details for delivery, or for formation of a collaborative users’ group. While the LOM model allows for the extension of the XML file, the risk is that of corrupting the standard’s interoperability. Information
idiosyncratic to a given application would be stored in the “extension” element, but would be unlikely to be understood by other applications. Extra overhead and possible confusion could be created by unnecessarily individualising a “standard”. By encapsulating application specific information in the object containing the standard file, this issue is avoided.

**Student_Agent**
The Student_Agent was configured to wait until the user requested the “find tutors” action. The agent prompted the user for a request file and then read it into the “Student Model” object. The agent set the request ID of the object and found the address of the Broker_Agent from the management system. It created a standard ACL message using the serialized Student Model and sent the message to the Broker_Agent. The Student_Agent then went into the “wait for response” state. The response was handled by writing it to an html file and creating a BrowserLauncher object that loaded the file into a browser for the user to see. The agent returned to the “wait-for-user-input” state.

**Broker_Agent**
Once the Broker_Agent was added to the environment and had registered its presence with the management system, it went into the “wait-for-request” behaviour. When it received the request from the Student_Agent together with the serialized student model, the Broker_Agent extracted the XML document from the model. It read the Yellow Pages of registered tutors for the locations of their descriptors. While filtering was not added at the point of reading the Yellow Pages, it could be added to a full scale application by comparing the tutor registrations with key elements of the Student_Request.

The Broker_Agent then found the LOM descriptors of the tutors and converted them to (w3c.dom) Java XML document objects. These were then converted to weighted document objects to enable prioritisation. A single method was implemented for comparing the content of the request with the content of the descriptor and assigning the descriptor a weight reflecting the quality of the match. This method could be expanded into an object with multiple methods for search operations allowing flexibility to trade accuracy for speed, to make searches that isolate particular elements or to make some elements more important than others. The descriptors
were then sorted, important aspects of their contents were extracted (like the location of the tutor) and the results embedded in html for return to the Student_Agent. The Xalan Java library was used to enable XPATH search techniques to be used on the Java XML document objects. The Broker_Agent sent the message and returned to the “wait-for-request” state.

User Access to Tutor

Once the User_Agent had saved the Broker_Agent’s response as a file and launched the browser to display the results (figure 15, below), the user was able to select a hyperlink that directed them to the location of the tutor. One of the ITS returned was web-distributed and the example Algorithm Tutor, being stand-alone, was downloaded using Web Start (screenshot of the download process is included as Appendix D.)

The resulting priorities reflected the needs of the student as expressed in the request. The student requested an algorithm tutor for tertiary level computing that was free. The resulting output from the four choices available returned the free version of the Algorithm Tutor before the one with a price. It returned a mythical tertiary level
bicycle strategy tutor (with price) next and the secondary level algebra tutor “Ms Lindquist” (free) last. The “response.html” is included as Appendix E.

<table>
<thead>
<tr>
<th>Tutor</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm Tutor – Academic Version</td>
<td>38</td>
</tr>
<tr>
<td>Algorithm Tutor</td>
<td>36</td>
</tr>
<tr>
<td>Bicycle Tutor</td>
<td>30</td>
</tr>
<tr>
<td>Algebra Tutor</td>
<td>22</td>
</tr>
</tbody>
</table>

**Figure 14: Prioritised result of ranking Tutors registered in the Yellow Pages.**

The particular prioritisation algorithm used took every text node in the request (with the exception of “LOMv1.0” – the source of predefined terms) and incremented the weight each time a match was found in the tutor’s LOM descriptor. While this gave no preferential treatment to individual requirements in the request, and was potentially a time-consuming approach, it proved to be able to make fine distinctions between the tutors. There should be no need to worry about response latency as one of the benefits of the use of agents is to liberate the user from having to be overly involved in the task. Users could be given a progress animation or be involved in other tasks and notified at an appropriate time.

**Summation**

The implementation demonstrated that it was possible to use agent technology to access a variety of Intelligent Tutoring Systems in a decentralised and accurately targeted manner. The system allowed for the prioritisation of results to match the user’s profile and was potentially expandable to a number of other behaviours related to student-centred computer-based education. The incorporation of accepted standards will allow real-scale implementation to be trialed.
TUTORS MATCHING YOUR PROFILE

The Algorithm Intelligent Tutor - Academic Version

Java Application 850000 Bytes

Required: JDK 1.4.1

The Algorithm Tutor is a stand-alone adaptive tutor that provides practice exercises in specific algorithms. Its modular nature means that the exercises can be regularly expanded. WebStart delivery notifies the user of regular updates. Search, sort and traversal algorithms are the base concepts for the exercises.

The tutor is intended as an adjunct to lectures and lecture notes, to enable a learner to reinforce their understanding of particular algorithms.

The software is distributed free provided an agreement exists between institutions and the author. Access to the software and its use is subject to the licence located at www.lawson.computas.edu.au/~kildare/tutor/licence.html

Priority weighting: 38

The Algorithm Intelligent Tutor

Figure 15: Browser containing prioritised tutor information.
Conclusion and Further Research

It has been demonstrated through the review of research and the performance questionnaire that Intelligent Tutoring Systems generally are viable in educational and performance terms. They are created as extensions of current educational theory and contribute to the psychological research about human learning. ITSs have repeatedly demonstrated their ability to scaffold the learning process and have been evaluated favourably against human tutoring performance.

Computationally, with the odd exception, these experimental tools were found to be performing within the bounds of what is considered to be satisfactory response latency. Research into which psychological factors are involved in determining the degradation of learning (due to slow response time) is sketchy and exact figures are non-existent. While this might present an area for further research, performances of ITS are such that research need not be considered from the perspective of this thesis.

New technologies have been shown to enable the distribution of Intelligent Tutoring Systems in a decentralised manner in keeping with recent student-centred educational practice. This form of distribution will help a large sector of the community in furthering their education. While there are educational materials other than ITSs that can be distributed using LOM standards and agent technology, the agent approach is particularly relevant to ITS because of the potential for expanding the use of agents to do more than search and retrieve. The potential for ITS to be deconstructed into component parts that mix and match is one area for exploration. Another is the ability to incorporate intelligent collaboration, using the agent mechanism to find students similarly engaged in a decentralised fashion.

Decentralised, user-centred search has been the domain of the search engine, but is not nearly as specific as using the LOM Descriptors and agents. “Algebra Tutorial” yields 200,000 results and “Algebra Tutor” yields “95,000” when checking Google. If one considers additional behaviours such as “purchase”, “enroll”, “sample” and
“share”, then the potential benefits of agents over search engines encourage further the exploration of agent-distributed ITSs.

It is possible to construct an ITS that retains many of the advantages of being centralised while remaining stand-alone, as exemplified by the algorithm tutor prototype. The Web Start facility allows for regular updates of the entire system with the user notified when new versions become available. The pedagogy and knowledge components can be modularized easily using XML definitions, so that new concepts and exercises can be added while reusing the engine.

There are a number of ways in which the agent distribution implementation could be extended. Superficially, the User_Agent could be given a graphical interface that provided functionality for creating the LOM compliant requests and for receiving the response from the broker. More importantly the development of an ontology for passing the messages, rather than serialized java objects, will enable the mobile and cross system benefits of agents to be fully realized. Further research into extending the behaviours of the agents (“purchase”, “enroll”, “share”, etc) would then be fruitful.

When considering the context of the deployment, questions of where the Yellow Pages directory sits, how it fits into a larger semantic web and just how it is updated also warrant further attention.
References


PROCTER, R. 2002. Usability 1: Principles and Guidelines. School of Informatics Edinburgh University. Available from:

RENKL, A., et al., 1998. Learning from Worked-Out Examples: the Effects of Example
Variability and Elicited Self-Explanations. *Contemporary Educational Psychology*,
23, pp. 90-108.

REYES, R. L. and GALVEY, C., 2000. Multimedia Intelligent Tutoring System For Context-


Environment with JADE - Java Agents for Distance Education Framework. *In: S. A.
CERRI, et al. eds. 6th International Conference, ITS*, Biarritz France and San
Sebastian, Spain, Springer -Verlag, pp 105-118.


SURAWEERA, P. 2001 *An Intelligent Teaching System for Database Modelling* Masters
Computer Science,University of Canterbury, Christchurch

http://www.eweek.com/print_article/0,3668,a=32912,00.asp [accessed Oct 2003]

System. *In: U. HOPPE, et al. eds. Artificial Intelligence in Education 2003*, Sydney,
Australia, Supplementary Proceedings - University to Sydney, pp 699-710.

TRAUSAN-MATU, S., et al., 2002. Ontology-Centred Personalized Presentation of
Knowledge Extracted from the Web. *In: S. A. CERRI, et al. eds. 6th International
Conference, ITS*, Biarritz France and San Sebastian, Spain, Springer -Verlag, pp
260-269.

May, 2003]

*In: S. A. CERRI, et al. eds. 6th International Conference, ITS*, Biarritz France and
San Sebastian, Spain, Springer -Verlag, pp 8-9.


### Appendices

#### Appendix A

**ITS Author Comments:**

<table>
<thead>
<tr>
<th>Name of Tutor</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry Explanation</td>
<td>The greatest complexity is the system's natural understanding module, not in its rule base. Speed has been a significant issue – this is somewhat of a trade-off between robustness and speed in natural language understanding.</td>
</tr>
<tr>
<td>LISTEN-reading tutor</td>
<td>Reading tutor not really suited for this questionnaire. Both interpreted and compiled, uses inheritance as its heuristic, rules are used for driving responses to events. Reading tutor not a rule processor - but has hundreds of classes and methods. [Response is given after the user reads in or selects &quot;go&quot;.] These questions seem to assume quite a lot about the form and behaviour of the tutor, and use terms like &quot;rules&quot; and &quot;content page&quot; that do not necessarily map cleanly to the Reading Tutor. I have tried to make reasonable mappings but the questions are not necessarily well defined for the Reading Tutor. I hope the order of magnitude estimates are usable in lieu of specific numbers.</td>
</tr>
<tr>
<td>Teatrix</td>
<td>This questionnaire doesn't suit Teatrix very well... 1. Our ITS is not a rule base system, e.g. as an expert System, although agents themselves are driven by simple rules. 2. There is not a structured task that can be evaluated, students goal when using Teatrix is to create a story by collaborating with other characters. The systems gives some narrative guidance to the students but doesn't explicitly build an user model that can be open and shared with the user...</td>
</tr>
<tr>
<td>ActiveMath</td>
<td>Feedback is of two types - local and global the local is immediate and the global on request. The number of rules depends on the strategy used. Java is the language used.</td>
</tr>
<tr>
<td>Story Station</td>
<td>Java. Big O not applicable. The main hardware constraint I struggled with was memory. Story Station loads a dictionary of word frequency information of about 11Megs, which it searches on user request. I have found that machines with less than 128M memory have trouble doing this. Extending the java max heap size to 500M helps.</td>
</tr>
<tr>
<td>REDEEM</td>
<td>Student model open depending on the teacher's requirements, similarly content size of display is determined by the teacher</td>
</tr>
<tr>
<td>AMICO</td>
<td>The language used to construct Amico is Java, so I have responded an &quot;interpreted language&quot; and by this way, Amico is a cross-over platform tool (I have tested this functionality)</td>
</tr>
<tr>
<td>KERMIT</td>
<td>Complex problems may take longer than 2 secs. Implemented in VB, currently being rewritten for porting to the web. Interface in Java and back end in LISP Backend server will be implemented using WETAS, the authoring tool</td>
</tr>
<tr>
<td>AutoTutor</td>
<td>The system is cross-platform insofar as it's a network application; the server runs on Unix, and the client runs on Windows. The data sent to the student is plain text. The system uses LSA for evaluation and not necessarily any set of symbolic rules; the number given is an approximate measure of the number of dialogue management rules executed based on the LSA cosine value of the student's response. Finally, this information is primarily applicable to the current Java-based version of AutoTutor; a new C# version is nearly complete.</td>
</tr>
<tr>
<td>DomoSim TPC</td>
<td>-</td>
</tr>
<tr>
<td>System</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ExcelTutor</td>
<td>The tutor is implemented using Visual Basic for Excel and interfaces with a LISP engine compiled in the form of an .exe file over Sockets.</td>
</tr>
<tr>
<td>SITS</td>
<td>Should be cross platform but not yet tested. CPU N/A. Order = number of Documents+number of Concepts.</td>
</tr>
<tr>
<td>AlgeBrain</td>
<td>Tutor built some time ago. Also provides feedback on request. (Processor speed estimated by Rob - &quot;certainly less than 1GHz&quot;)</td>
</tr>
<tr>
<td>SITTE</td>
<td>SITTE is a web-based assessment tool for Computerized Adaptive Testing. It can be used as a Diagnosis module inside an ITS architecture or even as a standalone tool.</td>
</tr>
<tr>
<td>Medtech</td>
<td>This system is no longer active. It was developed about 6-7 years ago. Speed was not an issue so no further development needed in this area. Java pages were about 200K</td>
</tr>
<tr>
<td>Stoic</td>
<td>very fast - once through the 60 rules. So no point optimising further.</td>
</tr>
<tr>
<td>Ms Lindquist</td>
<td>Question 11 is special to me. My system has 75 rules for the expert model and another 70 for the model of the expert human tutor that manages the dialog.</td>
</tr>
<tr>
<td>Disciple-COG</td>
<td>Authored using Disciple-RKF. The number of rules is really an estimate of 586 objects and features, 409 tasks and 419 complex task reduction and solution composition rules</td>
</tr>
<tr>
<td>Logic-ITA</td>
<td>Feedback immediate and on request. Logic tutor downloaded as a jar file and run on desktop heuristics used for validating the arguments – can be proven, no heuristics for processing student errors.</td>
</tr>
<tr>
<td>WADEIN</td>
<td>This program runs on the client.</td>
</tr>
</tbody>
</table>
## Appendix B

### Performance Survey Results:

<table>
<thead>
<tr>
<th>Latency</th>
<th>Frequency</th>
<th>Authoring Tool</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>authored</td>
<td>compiled</td>
</tr>
<tr>
<td>0 to &lt; 2 secs</td>
<td>14</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2 to &lt; 4 secs</td>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4 to &lt; 6 secs</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6+ secs</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TOTALS</td>
<td>20</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency</th>
<th>Frequency</th>
<th>Distribution Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authoring Tool</td>
<td>Standalone</td>
<td>Dist.Agents</td>
</tr>
<tr>
<td>0 to &lt; 2 secs</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>2 to &lt; 4 secs</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4 to &lt; 6 secs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6+ secs</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>TOTALS</td>
<td>20</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency</th>
<th>Frequency</th>
<th>Op System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authoring Tool</td>
<td>CrossPlatf</td>
<td>Windows</td>
</tr>
<tr>
<td>0 to &lt; 2 secs</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>2 to &lt; 4 secs</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4 to &lt; 6 secs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6+ secs</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>TOTALS</td>
<td>20</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency</th>
<th>Frequency</th>
<th>Optimisation(1=none; 4=Lots)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0 to &lt; 2 secs</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>2 to &lt; 4 secs</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4 to &lt; 6 secs</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6+ secs</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>TOTALS</td>
<td>20</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency</th>
<th>Frequency</th>
<th>Open Student Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>0 to &lt; 2 secs</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>2 to &lt; 4 secs</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4 to &lt; 6 secs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6+ secs</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>TOTALS</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>Latency</td>
<td>Frequency</td>
<td>Immediate</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>0 to &lt; 2 secs</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>2 to &lt; 4 secs</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4 to &lt; 6 secs</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6+ secs</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>TOTALS</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>num Rules</th>
<th>frequency</th>
<th>0 to &lt; 2 secs</th>
<th>2 to &lt; 4 secs</th>
<th>4 to &lt; 6 secs</th>
<th>6+ secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10 to 60</td>
<td>8</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>70 to 90</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100 plus</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>num Rules</th>
<th>frequency</th>
<th>authored</th>
<th>num modules</th>
<th>heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>1</td>
<td>3,4,5,7,7</td>
<td>2</td>
</tr>
<tr>
<td>10 to 60</td>
<td>8</td>
<td>2</td>
<td>1,1,1,2,4,5,7,20</td>
<td>1</td>
</tr>
<tr>
<td>70 to 90</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100 plus</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No.Modules</th>
<th>Frequency</th>
<th>0 to &lt; 2 secs</th>
<th>2 to &lt; 4 secs</th>
<th>4 to &lt; 6 secs</th>
<th>6+ secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 3</td>
<td>9</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4 to 7</td>
<td>10</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8 plus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No.Modules</th>
<th>Frequency</th>
<th>authored</th>
<th>num Rules</th>
<th>heuristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 to 3</td>
<td>9</td>
<td>2</td>
<td>0,0,10,50,60,60,150,200,500,800</td>
<td>1</td>
</tr>
<tr>
<td>4 to 7</td>
<td>10</td>
<td>2</td>
<td>0,0,0,0,10,27,50,100,124,200</td>
<td>2</td>
</tr>
<tr>
<td>8 plus</td>
<td>1</td>
<td>0</td>
<td>31</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Content Kb</th>
<th>Frequency</th>
<th>StandAlone</th>
<th>Dist.Agents</th>
<th>3-Tier Web-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 1</td>
<td>12</td>
<td>6</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2 to 17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18 to 65</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>66 to 400</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>401 to 500</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page Content Kb</th>
<th>Frequency</th>
<th>compiled</th>
<th>interpreted</th>
<th>num Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 to 1</td>
<td>12</td>
<td>3</td>
<td>9</td>
<td>0,0,10,10,27,50,50,50,100,124,200,500,800</td>
</tr>
<tr>
<td>2 to 17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18 to 65</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>0,0,60,60,150</td>
</tr>
<tr>
<td>66 to 400</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>401 to 500</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0,3,1,200</td>
</tr>
</tbody>
</table>
Appendix C

Student_Agent State Change Diagram

Broker_Agent State Change Diagram
Appendix D

Downloading the Algorithm Tutor Using Web Start

The Algorithm Tutor

Hello. Welcome to the Tutor Download Page.
Launch the application
Appendix E

Response.html: the result returned by the Broker_Agent.
can be downloaded at no cost. Online registration of a username is necessary but no personal data is required. <a href="http://www.algebratutor.org">Find</a> <br/><b>Priority weighting:</b> 22
Appendix F

Example DTD for Algorithm Tutor

<!-- example dtd demonstrating how to define exercises for decentralised ITS -->

<!-- specific algorithm to practise -->

<!ELEMENT algorithm (title, question+, concept)>
<!ELEMENT title (#PCDATA)>

<!-- multiple questions possible -->
<!ELEMENT question(problem, stage+)>
<!ATTLIST question IDNumber CDATA #REQUIRED>

<!ELEMENT problem(#PCDATA)>

<!-- each question can have levels of difficulty called "stage" -->
<!ELEMENT stage(step+, answer+)>
<!ATTLIST stage stageNumber CDATA #REQUIRED>
<!ATTLIST stage difficulty (general-simplest | mixed-easy | mixed-hard | specific-hardest) #REQUIRED>

<!ELEMENT step(hint+, feedback+, #PCDATA)>
<!ATTLIST step stepNumber CDATA #REQUIRED>
<!ATTLIST hint stepNumber CDATA #REQUIRED>
<!ATTLIST hint level (general-hardest | mixed-hard | mixed-easy | specific-easiest) #REQUIRED>

<!ELEMENT feedback(#PCDATA)>
<!ATTLIST feedback stepNumber CDATA #REQUIRED>
<!ATTLIST feedback level (general-hardest | mixed-hard | mixed-easy | specific-easiest) #REQUIRED>

<!ELEMENT answer('#PCDATA)>
<!ATTLIST answer stepNumber CDATA #REQUIRED>

<!-- note the knowledge has been loosely described here in a recursive form -->

<!-- Will recursion be viable when retrieving information using XPATH?? -->

<!-- common errors made by students plus confidence level in their understanding is encodable -->

<!ELEMENT concept((description, concept*, relationship*, buggyRule*, confidence)>
<!ATTLIST concept type (precondition | process | postcondition) #REQUIRED>

............<!-- incomplete -->............

...
## Glossary of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>Third Generation mobile device</td>
</tr>
<tr>
<td>ACL</td>
<td>Agent Communication Language</td>
</tr>
<tr>
<td>ACT-R</td>
<td>Atomic Components of Thought -Rational</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>AIED</td>
<td>Artificial Intelligence In EDucation</td>
</tr>
<tr>
<td>CAI</td>
<td>Computer Assisted Instruction</td>
</tr>
<tr>
<td>CBM</td>
<td>Constraint Based Modelling</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Common Object Model</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>ECA</td>
<td>Enterprise Collaboration Architecture</td>
</tr>
<tr>
<td>FIPA</td>
<td>Foundation for Intelligent Physical Agents</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HCI</td>
<td>Human Computer Interface</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent tutoring System</td>
</tr>
<tr>
<td>JADE</td>
<td>Java Agent DEvelopment framework</td>
</tr>
<tr>
<td>JIT</td>
<td>Just In Time compiler</td>
</tr>
<tr>
<td>KQML</td>
<td>Knowledge Query Mark-up Language</td>
</tr>
<tr>
<td>LOM</td>
<td>Learning Object Metadata</td>
</tr>
<tr>
<td>NLU</td>
<td>Natural Language processing Unit</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Calls</td>
</tr>
<tr>
<td>SPML</td>
<td>Service Provision Mark-up Language</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TCP/UDP</td>
<td>Transport Control Protocol/User Data Protocol</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>VLAN</td>
<td>Virtual Local Area Network</td>
</tr>
<tr>
<td>WCL</td>
<td>Wireless Communication Language</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>'eXtensible Mark-up Language</td>
</tr>
<tr>
<td>XPATH</td>
<td>XML Path language</td>
</tr>
</tbody>
</table>
Declaration

To the best of my knowledge and belief,

this thesis contains no material which has been accepted for the award of any other degree or diploma in any tertiary institution, and this thesis contains no material previously published or written by another person except where due reference is made in the text.

Name: Robert Andrew Kildare

Signature:

Date:
Abstract

Recent changes in technology make it possible to distribute Intelligent Tutoring Systems (ITs) so that students who are not tied to centralised systems can benefit from adaptive software. Many social benefits can flow from wider access to ITs. An extensible architecture is described for deploying Intelligent Tutoring Systems in a student-centred manner. Large scale research projects have been in place, some for over 15 years, validating educational strategies, experimenting with variations on typical tutor architectures and using a range of distribution methods. Authoring tools now exist which trade the depth of intelligence of their progeny in order to make ITs accessible to less technically skilled teams of educators. The survey of computational performance in this study shows rapid input-response latency to be the norm. It is demonstrated that ITs are generally educationally and computationally viable software and ready for wider distribution.
Acknowledgements

I am indebted to the following people for their support in the creation of this thesis:
my supervisor, Robyn Gibson, for her advice and encouragement, Nicole Clark and Mark Hepburn for their corrections and comments, and the support staff and honours co-ordinators for providing the infrastructure to enable the research to occur.

Further afield I would like to thank the participants in my survey of Intelligent Tutoring System performance. The organisers of the AIED2003 conference provided a travel bursary, volunteership and the opportunity to make the survey available to the AIED community. The conference was an invaluable stimulus to the research.

Shaaron Ainsworth, Christopher Eliot, Nancy Alvarado, Brent Martin, Kalina Yacef, Neil Heffernan and Beverly Park Woolf deserve thanks for their patience with my email and also Vania Dimitrova and Julietta Vassileva for their insightful guidance.