Knowledge Representation and Acquisition for Intelligent Tutoring Systems

2003

Steven Linton **Supervisor: Dr Brent Martin**

Abstract

Intelligent Tutoring System (ITS) Authoring Tools have successfully been able to mechanise the creation of the intelligent components of the ITS. However, each of these components have a high level of coupling with the domain model, which is used to store the domain knowledge. This domain model must be complete and accurate otherwise the tutor will not perform effectively. The process of creating the domain model remains problematic due to the large amount of time that must be invested. This research investigates methods for simplifying and automating the process of creating the domain model. The focus is on improving the architecture and representation using XML and the Model-Driven Architecture (MDA) to aid portability and reuse.

Acknowledgements

I would like to thank Dr Brent Martin for his invaluable feedback and guidance throughout the research and my parents for enduring the torturous proof reading of this report.

Table of Contents

1 Introduction 4
1.1 Summary 4
2 Background 6
2.1 Knowledge Classification 6
2.2 Intelligent Tutoring System (ITS) 6
2.2.1 The Domain Model 7
2.2.2 The Student Model 7
2.3 Domain Modelling 7
2.3.1 Constraint-Based Modelling (CBM) 8
2.3.2 Buggy Constraints 8
2.4 XML Technologies 9
2.5 Model-Driven Architecture (MDA) 9
3 Related Work 10
3.1 ITS Authoring Tools 10
3.2 Constraint Representation 11
3.3 Ontology Engineering 12
4 Knowledge Representation 13
4.1 Domain Ontology 13
4.2 Domain Concepts 14
4.2.1 Open Student Model 16
4.3 Domain Definition 17
4.3.1 Design Considerations 18
4.3.2 Namespaces 19
4.3.3 XML Schema Definition 19
4.4 Domain Knowledge 19
4.4.1 Modal Constraints 20
4.4.2 Feedback 21
4.4.3 XSLT Definition 22
4.5 Courseware (Problems and Solutions) 22
4.5.1 Question Generation 23
4.5.2 Validation 23
4.5.3 XML Definition 24
5 Knowledge Acquisition 25
5.1 Primer 25
5.2 Programming by Example 26
5.2.1 The Algorithm 26
5.2.2 Programming by Demonstration 28
5.3 Transformation Approach 28
5.3.1 Analysis 29
5.3.2 Limitations 30
5.4 Inquiry-Based Approach 30
6 Future Work 32
6.1 Build a MDA ITS 32
6.2 Evaluate Transformation Approach 33
6.3 The Semantic Web 33
7 Summary 35
8 References 36
9 Appendix 39
9.1 ER Domain Schema 39

1 Introduction

Intelligent Tutoring Systems (ITS) are educational systems that aim at providing one-to-one tuition of students. The student interacts with the system by solving problems that are selected based on that student's ability. A domain model is used to store an understanding of the instructional domain, which is to interpret a student's solution and track their skills. Due to the large number of complex components comprising an ITS, Authoring Tools have been developed to aid their construction. This provides most of the intelligent components, yet the construction of the domain model remains notoriously difficult.

Efforts to simplify this problem have been primarily involved with the definition of domain modelling languages. These languages define the process of how a problem is interpreted. Of interest in this research is the Constraint-Based Modelling (CBM) approach which models knowledge using independent domain rules. These rules have been shown to be faster to elicit than other knowledge elements of equivalent approaches [1]. Estimations place the time to create a single constraint at approximately one hour. This seems fast, yet in an average to large domain there can be between 100 to 500 constraints, which results in many weeks work to create a new ITS.

It is believed that this is one of the main reasons that ITS's have not be widely integrated into Schools and workplace training schemes [2]. Another reason cited for this lack of ITS acceptance is in the lack of configurability [3] and reuse [4]. This is an issue that has been addressed to some extent by ITS Authoring Tools, however these tools are often constricted to fixed domain modelling techniques and representations. This is caused by the high level of coupling of the domain model with the other ITS components.

The goals of this research are two-fold. The first goal is to demonstrate knowledge acquisition techniques that can aid in the construction of new domain models. To achieve this goal effectively a detailed analysis of the domain model must be performed. The architecture of the ITS and the knowledge representation has a strong influence on how efficiently the domain knowledge can be acquired. Therefore the second goal is to illustrate how the domain model, for use in ITS Authoring Tools, can be designed to enable automated knowledge acquisition, higher speed system development, portability and the integration of independent domain models.

1.1 Summary

The following paper is organised as follows. Section 2 provides the background knowledge of ITS systems and the technologies that are proposed to optimise them. This is followed by the related work that has been performed in the two areas that form the major component of this report; knowledge representation and knowledge acquisition. Knowledge representation is discussed in Section 4. Here methods are described for the simplification and extension of the domain model design. The standard CBM domain model uses constraints as the only representation. The extension proposed, represents the domain using three models: the domain concept model, the domain definition and the domain knowledge. The domain concept model captures the high-level, abstract concepts that the ITS is attempting to teach the student. The domain definition is a low-level, syntactic definition of the domain that provides a language to define ITS courseware and knowledge. The domain knowledge is composed of constraints, or in a Model-Tracing (MT) tutor, the production rules and bug library. By defining each of these components using XML and structuring it with the Model-Driven Architecture, the domain model is portable and platform-independent. These properties lead to the future of ITS's which integrate many domains across a distributed web-based environment.

Based on this new knowledge representation, three knowledge acquisition approaches are discussed in Section 5. The first approach is programming by example; this treats the ITS as a student, supplying it with an ideal solution and having it permute its definition to produce new solutions. The new solutions are validated by the domain expert. The system infers the effect of alterations and generates a constraint that captures that state change. The second approach exploits the nature of the XML domain definition. By using pre-defined templates, it searches the domain definition for patterns that indicate constraints common throughout different domains. Finally this transformation approach is augmented by a high-level inquiry system. This aids the domain expert in mentally exploring the domain knowledge by asking questions based on observations of the domain model state.

2 Background

This section provides a brief overview of the Intelligent Tutoring field and a motivation for the research component of this report. It explains the role of the domain knowledge and representation, and the ITS components that interact with it.

2.1 Knowledge Classification

Although elementary, an investigation of ITS's must begin with an understanding of what intelligence is and how it is presented in an ITS. An ITS has knowledge of an instructional domain and is able to provide feedback that reflects this. ITS's typically select a knowledge base that represent either declarative or procedural knowledge. For the purposes of this research, declarative knowledge in the form of heuristic rules (constraints) has been selected (discussed further in Section 2.3.1).

Knowledge of the domain is not the only consideration. Constraints do not encapsulate the concepts of a domain. Concepts in this situation are considered to be the general ideas that the student must know to understand the domain. ITS's that represent domain concepts are able determine how well a student is progressing in a specific area, and when to move to the next. Finally, the system must have a syntactic definition of the domain to understand how to parse and process it.

2.2 Intelligent Tutoring System (ITS)

The ITS provides a student with interactive and adaptive tuition with intelligent feedback. Interactivity is the key function, studies have shown that students learn best when they are presented with a problem that they must solve manually. ITS's typically provide a large repository of problems for the student to work through. The path that they take is generated adaptively based on the student's performance. While a student solves a problem the ITS records statistics that can be used to indicate the students level of skill. These statistics are stored in a student model, which encapsulates the domain concepts. The adaptability is facilitated through teaching strategies, which control the solution feedback and the sequencing of the course. There are a vast number of teaching strategies and sometimes many are implemented within a single system. The alternative to ITS's are Computer-Aided Instruction (CAI) systems. These have no intelligent component and simply present the student with static courseware.

Modern ITS's and ITS Authoring Tools are component-based to reduce the complexity of their implementation. The component diagram in Figure 1 shows each of the typical components for a CBM tutor. There are three components and three repositories. Each repository has no processing capability this is provided by the ITS components. The courseware for an ITS typically consists of problems and their ideal solutions. Many ITS's do not include instructional material. ITS's do not aim to replace human tutors, rather to provide a supplement to regular tuition to reinforce the concepts that are taught [5]. Of particular interest for this research is the domain model, which stores the domain knowledge required to make the ITS intelligent, this is explored further in the next section. The constraint-based modeller is responsible for taking a Student Solution and interpreting its validity using the domain knowledge. Constraints are only one way to model the domain, this is explained in more detail in Section 2.3. The modeller is what differentiates an ITS from an Adaptive Hypermedia

Tutor, that contains a Student Model but has no facility for proving intelligent feedback for student solutions. The pedagogical module is responsible for the teaching strategies.

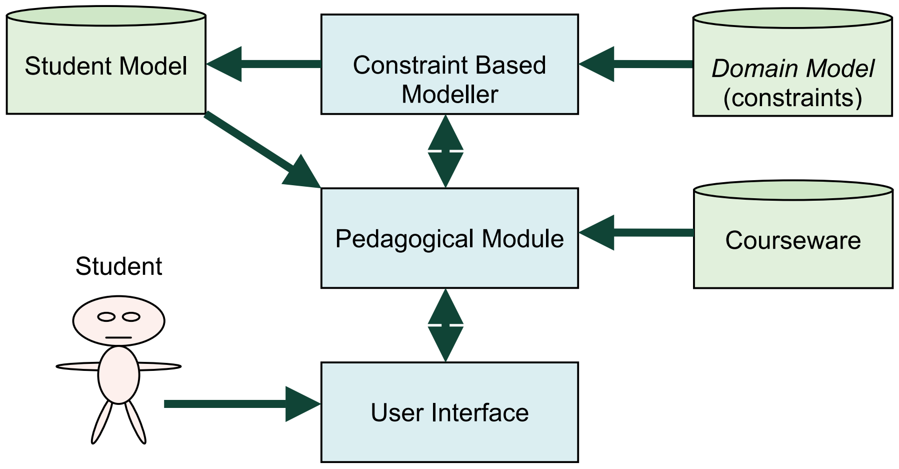

Figure 1 : The ITS architecture of KERMIT [6]

2.2.1 The Domain Model

Whether or not it is explicitly stated, all ITS's must have a domain model. The domain model stores the knowledge required to interpret the instructional material. This knowledge is formed from low-level rules or heuristics that describe the detail of domain concepts. This knowledge is represented in a language that is understood directly by the other components of the system. Specifically, its representation strongly influences the approach to student modelling and the pedagogical processing of the system. The domain model is not inherently intelligent; the intelligence is a result of how the domain model is used throughout the system.

2.2.2 The Student Model

The student model stores historical data, tracking a student's progress through the courseware. This data is used to generate behavioural statistics that model the knowledge of the individual student. The student model has been the primary focus of most ITS research as it provides most of the ITS's intelligence. The uses include customised feedback, question selection and adaptive structuring of the courseware.

There is a rich variety of student modelling approaches, yet each is dependent on the content of the domain model knowledge. ITS architecture designs typically show the student model as independent of the domain model [6], yet in practice the student model stores its data using the domain model. The student model is either a subset of the domain model or encapsulates it entirely, while overlaying its own statistics. This coupling increases the system complexity as changes to the domain model must be reflected in the student model. This flaw remains a restriction of present ITS designs.

2.3 Domain Modelling

Domain modelling encapsulates the representation and processing of domain knowledge. As the domain is the core of the ITS, the method used to model it is critical in defining the operation of the system. Much research has been performed into the definition of appropriate domain modelling techniques. Two important techniques are Constraint-based modelling (CBM) and Model Tracing (MT), which represent declarative and procedural knowledge respectively. Each approach has strengths and weaknesses depending on the type of domain they are modelling. CBM is more effective for modelling open-ended domains such as languages or design notations and practices, whereas MT is more suited to well-defined domains such as mathematics or physics. A comparative analysis of the two techniques is provided in [1]. The most important observation made, for the context of this research, is that constraints are both easier and consistently faster to manually generate. The Constraint-Based approach is the focus for knowledge representation and acquisition techniques presented in this research.

2.3.1 Constraint-Based Modelling (CBM)

Constraints capture the declarative knowledge of the domain. The concept of modelling the domain knowledge using constraints was introduced in 1993 by S. Ohlsson [7]. It should be noted that domain knowledge and student knowledge are terms often used interchangeably, as Ohlsson's paper does. However, they do represent quite different concepts. Constraints are independent, modular heuristics that capture the domain facts. A set of related facts form domain concepts. They are categorised as syntactic and semantic. Constraints use pattern matching between the ideal solution and the student solution. Therefore evaluation of a solution in a CBM tutor should be performed when the student has completed the problem. Each constraint is structured with a relevance and satisfaction condition. Only constraints that are relevant are applied to the problem, those that are not satisfied produce some feedback. The following example shows a constraint, represented in plain English, which checks that "all *Entities* have unique *names*[1]".

Relevance Condition: This is an Entity of the Student Solution.
Satisfaction Condition: The Student Solution contains no other Entity with that name.

An additional limitation of constraints is that often they are engineered to only consider the relationship between the ideal and student solution but not the problem question. In open-ended domains, where the solutions are of sufficient size, this is generally appropriate. However, in more formal domains, the question is of critical importance for inferring whether a solution is correct. As an example, a mathematics tutor may pose the question, "What is 1 + 1?" The ideal solution is *2* yet if the student answers *1*, based on ideal and student solution only, there is no way for the system to determine that the student has multiplied the parameters. In the paper [1] that compares MT and CBM tutors the argument is that declarative domains are more suited to MT. A solution to this problem is explored in Section 4.5.

2.3.2 Buggy Constraints

Ideally, constraints only identify the correct states of the solution; they do not search for what the student has done wrong. Constraints that do this are known as buggy constraints. Buggy Constraints are derived from normal constraints and the feedback they provide is therefore too detailed; it tells the student specifically their error which is not constructive to a learning environment [8]. By capturing student errors the

[1] This is a common example used throughout the document and was taken directly from KERMIT.

system is more accurately able to model a student's knowledge, yet these constraints should not be used to provide feedback directly to the student.

2.4 XML Technologies

XML (eXtensible Markup Language) [9] and its many related technologies are an important theme throughout this research. It is used as the representation language for the domain model and courseware. The following is a brief overview of the various relevant technologies and what they offer.

XML is designed for the standardisation of Internet documents that contain structured information. XML was introduced in 1997 and was quickly accepted by industry. It is highly portable; an XML document can be processed on all major platforms and web browsers. The XML syntax is defined using sets of tags and structural relationships to describe a document. The syntax of an XML document is described and constrained by an XML Schema [10]. Semantics are not explicitly defined for an XML document; this is facilitated through an XSLT (eXtensible Stylesheet Language Transformations) [11] document. XSLT reads through an XML document searching for tags of interest and transforms the information they contain to another format such as a web page or another XML document. XPATH [12] provides the pattern matching capabilities of XSLT.

2.5 Model-Driven Architecture (MDA)

All of the simplifications, enhancements and extensions can be tied together through the Model-Driven Architecture (MDA) [13]. The MDA is a software engineering design principle introduced by the OGM (Object Management Group) in 1997. It relies on the concept of platform-independent model (PIM) and platform-specific model (PSM). All of the information should be specified at the platform-independent level using XML. XSLT transforms can then be used to convert the PIM to a PSM, which can subsequently be run.

The ITS domain Ontology (Section 4.1) represents the PIM, whereas the domain knowledge is a PSM. In this sense the platform is considered to be the specific domain modelling technique used in the ITS. There are currently no tutors that implement two radically different domain modelling techniques simultaneously. A second platform is the low-level representation used for the domain model and courseware. WETAS uses a derivative of LISP for which it has its own parsing engine. The research presented here uses the high-level XML language for the representation; this overlays the implementation-specific platform. The greatest benefit of introducing MDA to ITS's is that the integration is at a high-level; the existing implementation could still be used without any major modification. ITS's and Authoring Tools can benefit greatly by structuring the domain model and courseware with the MDA.

3 Related Work

This section describes some of existing research that has been performed into knowledge representation and acquisition. There are a large range of concepts that are covered in this research including ITS's, knowledge engineering, ontological engineering, machine learning, XML and the MDA. Each of these fields have vast amounts of literature associated with them. Only the most important related work is presented in this section. Knowledge representation research is most involved with the development of ITS Authoring Tools and the integration Ontologies. However, knowledge acquisition has seen few practical and successful approaches when applied to ITS's.

3.1 ITS Authoring Tools

Creating a new ITS from scratch is a major task involving many thousands of hours to build the various components required for a functioning system. Fortunately, ITS authoring tools and shells have been created that supply most of the complex components and provide a standardised approach to creating an ITS. The goal of the ITS Authoring Tool is to aid development by decreasing the effort and skill required, to aid the designer to consider the domain, to support good design principles and enable rapid prototyping [14]. Authoring Tools such as WETAS [15] are complete enough that only the domain model must be completed to create a new ITS. Even so, the effort required to build the domain model knowledge-base is very high, with estimates anywhere from several hundred to several thousand work hours, depending on the modelling technique [1] and size of the domain.

The main difficulty still facing ITS authoring tools is that they are not generic and there are no standards. Each defines a proprietary structure and domain representation. In [14] T. Murry describes seven categories for the classification of ITS Authoring Tools by type. These are:

- Curriculum Sequencing and Planning
- Tutoring Strategies
- Device Simulation and Equipment Training
- Domain Expert System
- Multiple Knowledge Types
- Special Purpose
- Intelligent/Adaptive Hypermedia.

WETAS [15] is a domain expert type ITS Authoring Tool, and uses the constraint-based approach of domain modelling. WETAS is both the tool for creating an ITS and a shell for distribution across a web-based interface. It uses the Allegroserve Web Server and is capable of hosting many domains and sub-domains simultaneously. Constraints are implemented using the Allegroserve Lisp language, which means that ITS domain experts must be familiar with this language. Some effort has been made to simplify this by identifying and limiting the constructs to those that are required to create constraints [16]. WETAS provides the infrastructure and the intelligent components required to interpret the domain model. The domain model and courseware are effectively the only aspects that need to be constructed. The authoring interface for this includes a visual Ontology that allows the designer to construct a simple diagram to illustrate the structure of the domain. This aids the domain expert with their mental exploration of the system. WETAS has been shown to be effective [17] through four ITS's that were implemented with it: SQL-Tutor , KERMIT, NORMIT and LBITS.

REDEEM [18] is a widely successful ITS Authoring Tool, which identifies the fact that educators are most concerned with the input of the courseware. They wish to have full control of how the system represents their instructional material. The power of REDEEM is in its ability to take existing computer-based instructional material and use that as the domain model. It uses its built-in knowledge, student model and supplemental domain knowledge defined by the domain expert, to provide adaptive sequencing of the courseware. In contrast to WETAS, the interface is aimed at the layman ITS user, where the courseware is structured using friendly Graphical User Interfaces. Unlike WETAS though, there is no facility for problem solving and is therefore not as effective at tutoring students. Formative evaluations have shown that REDEEM does not perform any better than an equivalent CAI tutor, and in some cases was even worse [19]. This highlights that intelligence is an important part of an ITS and that user acceptance should not take precedence over functionality.

Demonstr8 [20] is another ITS Authoring Tool, which targets MT domain modelling based on Anderson's ACT-R theory [21]. In MT ACT-R the domain model is represented using production rules, which are statements with a condition and action. Concepts are known as skills, which are a set of production rules. ACT-R has a clear line between declarative and procedural knowledge. In Demonstr8 the declarative knowledge is in the form of a data definition that is used to describe the problem and working memory structure. The procedural knowledge (production rules) supplies the intelligence and like WETAS, is defined with LISP. The processing capabilities are provided by the Authoring Tool. The user must only define the domain structure, the interface and the production rules. The strength of Demonstr8 is in its Program by Demonstration method for acquiring production rules. The domain expert takes the place of the student and trains the system using real examples. The system extracts production rules by observing and inquiring with regard to the expert's actions. An algorithm also exists for the abstraction and generalisation of constraints. Unfortunately, Demonstr8 did not make it further than the prototype stage. It was found that production rules could not be automatically extracted without considerable background domain knowledge as the generalisation algorithm made too many mistakes. Others are looking to address these issues by reducing the level of automation but increasing the robustness and generality [22].

3.2 Constraint Representation

One of the advancements to CBM ITS presented by B. Martin in his doctorate thesis was the WETAS constraint language [16]. The language is a derivative of LISP and consists of three fundamental functions used for pattern matching: MATCH, TEST and TEST_SYMBOL. These statements have been sufficient to describe constraints for SQL (SQL-Tutor [23]), ER (KERMIT [6]) and elementary English (LBITS [16]) tutors. An early version of SQL-Tutor used pure LISP, which resulted in a complex and undisciplined domain model definition. Functions could be created that captured specific aspects of the domain. The WETAS constraint language successfully introduced a structured approach, increasing the maintainability of the constraint set.

The aim and result of this new constraint definition language was to make the domain executable, so that questions and solutions could be automatically generated [17]. While automatic generation of questions is not a focus of this research, it uses similar concepts that are relevant to domain model generation. The process of generating problems and solutions was to select a group of constraints that encapsulate the concept that the student needs to learn. Constraints represent low-level forms of the domain knowledge; it was observed that fragments of the domain could be extracted from each constraint and placed appropriately within the problem definition. This problem was them put through a problem solver which generated the solution. Finally, the question was converted to natural language and presented to the student. The primary difficulty with this approach is that creating a problem solver is problematic in most domains. This is why constraints exist, as problem solvers are too difficult to build and to generalise to all domains within an ITS Authoring Tool.

3.3 Ontological Engineering

The aim of the Ontology is to define an explicit representation for the conceptual domain component, which few ITS's attempt. It is argued that to create intelligent systems there must be a declarative definition of what they know. The system defines terms and concepts that are used to describe the domain expert and perform problem solving independently of the domain. This gives the ITS the ability to analyse and process a knowledge-based system. In the paper {Mizoguchi, 2000 #58} the Ontology is described as having three levels:

1. A structured collection of terms that define a conceptual hierarchy of the domain.
2. A formal definition of the domain concepts and relationships through constraints and axioms. This is to avoid ambiguity and make the domain machine readable.
3. An executable version of the domain.

The paper {Mitrovic, 2002 #33} describes the M-OBLIGE model, which implements this architecture using XML and Protégé-2000 for the representation. It describes how this model can be applied to integrate KERMIT and SQL-Tutor using local Ontologies for each domain. Each of these local Ontologies are integrated into an external Ontology, which is used to interact with other ITS's. It is hoped that this infrastructure will suggest a framework towards the unification and interoperability of multiple tutors.

4 Knowledge Representation

When considering the simplification and automation of creating an ITS, the representation of the domain model is of critical importance. The representation is a complex problem as it can affect every component within the ITS. There are two ways to approach increasing the efficiency for knowledge representation in ITS's. The first is to investigate new domain modelling techniques that will allow for simpler generation of the domain. This also requires the consideration of how each of the ITS components will use the new domain to process students interactions with the system. CBM has been selected as the focus for research as its representation is consistently defined throughout CBM research [5, 7], it stores declarative information and has been proven to be effective through evaluations of real systems [5]. An initial investigation showed that the modular nature of constraints make them ideal for knowledge acquisition as each knowledge element is independent. It also indicated that this modularity could be extended to make the domain model platform independent. This section describes how the MDA can be used to achieve this and how the domain model can be optimised for more efficient use throughout the system, to enable automated acquisition.

The domain model in a CBM tutor is typically represented solely by using constraints. As a result the ITS contains a very specific knowledge of the domain. Higher level concepts are lost or are supplemented without integration. A complete domain model must also include a model of the domain concepts, and an explicit representation or definition. The domain concepts and definition can be implemented independently of ITS components and as such, are generalised to form an Ontology. This Ontology is interchangeable between systems, be they an ITS or otherwise. To enable this portability while maintaining integration with the ITS, XML is suggested as the physical representation for the Ontology, domain knowledge and associated material.

The physical representation of the domain model for CBM tutors is typically implemented using a functional language such as LISP [16, 24], commonly used for AI and logic applications. While such languages are extremely effective, both in their simplicity and their performance, they do have an inherent flaw. Domain models are not portable without great effort due to the syntax of the language and the lack of standardisation between ITS research communities. This section demonstrates how XML can mimic the functions of the WETAS domain language (presented in Section 3.2), while maintaining portability and integration between components. It is further used in Section 5 to enable effective XML based domain learning algorithms.

4.1 Domain Ontology

The concept of a domain Ontology is relatively new in the field of Intelligent Tutoring [25, 26]. However, they have received much development in other fields [27] and is one of the key technologies of the Semantic Web (Section 6.3). The formal definition of Ontology is the "*specification of a conceptualisation*". It provides a definition of formal domain vocabulary and allows for knowledge sharing and reuse [28]. A visual Ontology tool is provided as a module for the WETAS authoring tool. This allows domain experts to model domain concepts and create relationships with associated constraints. Like any effective notation, this has a profound effect on both the general understanding of a domain and its relationships. Using the Ontology simply for conceptual visualisation is limiting its power, as the domain structure (Figure 3) is not represented.

The domain models illustrated in Figure 2 shows how the domain Ontology captures two of its aspects: the *domain concepts* (Section 4.2) and the *domain definition* (Section 4.3). The *domain knowledge* (Section 4.4) is considered external as it is at a different level of abstraction. While the semantics captured in the domain knowledge model are platform independent, their representation is dependent on the domain modelling platform (in this case CBM). The most important concept of the Ontology is that it is platform-independent and generic. The Ontology must not require or imply any method for processing it. This notion of platform-independence comes from the Model-Driven Architecture (Section 2.5). It is this nature that allows the Ontology to be integrated with other tutors; the Ontology content can be merged as each domain is syntactically independent. This requires an effective interface between the domain Ontology, domain knowledge and ITS components.

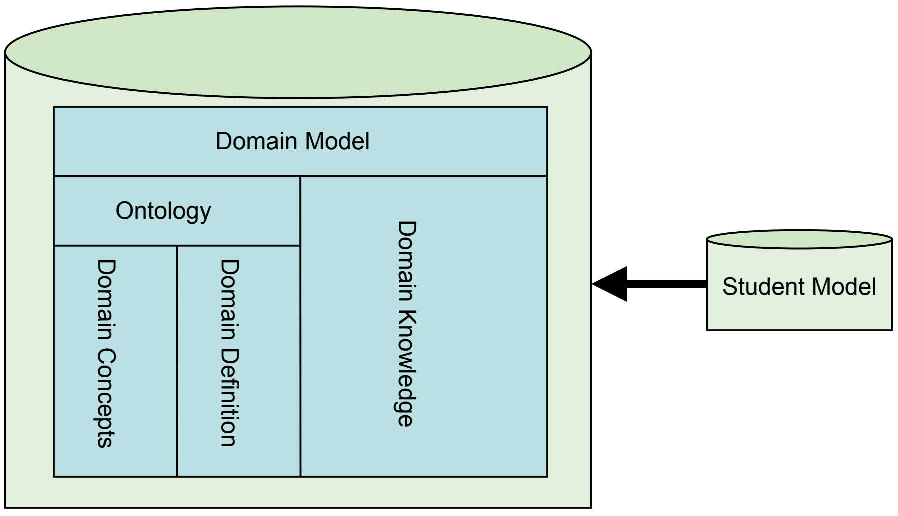

Figure 2 : Decomposition of the proposed Domain model, illustrating that the Ontology is comprised of the Domain Concept Mode and Domain Definition, but not the Domain Knowledge.

The Ontology also simplifies the maintenance of the domain model [29]. Many domains are under constant external development (e.g. UML or XML) and would provide a major challenge to maintain as ITS. Without a formal definition for the domain and its concepts it is difficult to track changes. In a traditional system a change required reinvestigation of the entire domain model to determine what is no longer relevant and where the required changes must be made. This is a tedious process and prone to errors. With the Ontology, changes are propagated throughout the system, any constraints or courseware that relied on it will no longer pass validation against the schema, immediately identifying the effected material.

4.2 Domain Concepts

The domain concepts are the highest level representation of the domain model (Figure 3). Domain concepts are a direct abstraction of the domain knowledge facts and semantics, yet are not explicitly present within either the domain definition or the domain knowledge. Modelling of domain concepts is a subjective process as it is dependent on the domain modeller, who must attempt to model a combination of the domain semantics and domain definition abstractions. Although a finalised concept model is tightly defined, using some modelling notation, it can be the most difficult design aspect of the domain model. One of the key purposes of the domain concepts model is for the categorisation of domain knowledge elements.

The concept model more closely resembles the instructional topics and sub-topics than the explicit domain structure represented in the domain definition. Unlike the Constraints, questions and solutions, the domain concepts are not represented in terms of the domain definition (as explained in Section 4.3). In most cases abstract concepts are not captured explicitly in the syntax of a domain. Yet, this lack of integration does not cause incompatibility within the system; these linkages must be created manually by the domain modeller. It is not yet possible to optimally infer the conceptual structure of domain concepts automatically, although there are efforts to accomplish this [30]. As the domain concept model is subjective, less importance is placed on making it portable.

The domain concepts are represented using a semantic network structure. Nodes represent knowledge concepts, while the links represent the relationships and interactions between them. Semantic networks are used to describe declarative and procedural knowledge. There are many notations for the visualisation and structural definition of semantic networks. Although not a formal notation, UML Class diagrams have been widely accepted [31, 32] for semantic network visualisation. An example of how UML can be used to represent a semantic network is illustrated in Figure 3. This shows the conceptual layout of a simple ITS for tutoring use of Adjectives. UML is appropriate for this research as it has an associated XML representation known as XMI [33]. This can be easily transformed using XSL for subsequent use throughout the ITS components. Another benefit of UML is that it effectively represents semantic network relationships, which can be lacking in the ER notation (a conceptual data model more commonly used for semantic networks). Critical semantic network relationships and their equivalence in UML notation are: is-a ⇔ inheritance, uses ⇔ association and has ⇔ aggregation.

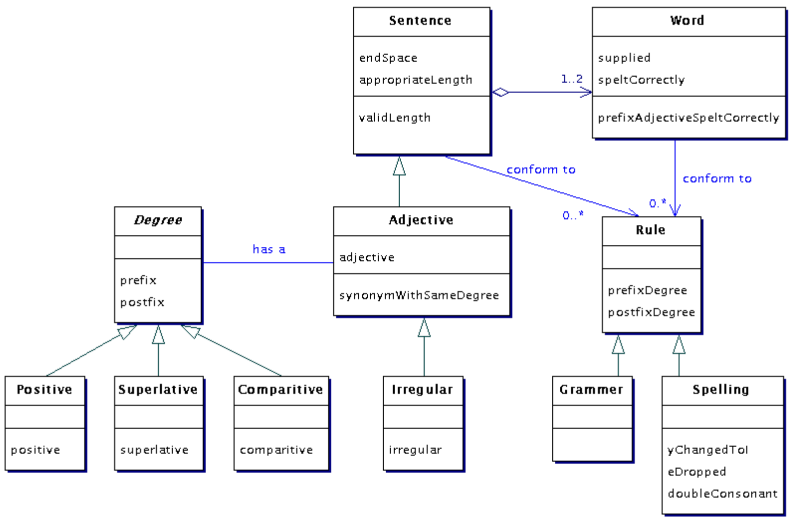

Figure 3 : A Domain Concept model for an Adjectives tutor.

The key purpose of formally representing the domain concepts is that they act as a classification medium for the low-level domain knowledge. This improves the opportunities for both the Open Student Model and Question Selection ITS components.

4.2.1 Open Student Model

The Open Student Model is one of the ITS components that benefits from the explicit inclusion of the domain concept model. The purpose of an Open Student Model is to illustrate to the student what concepts they understand and remain to be learned [34]. This information is derived from the Student Model, which itself is a derivative of the domain model. The student is presented with a visual representation of the domain concepts with contextual information inlaid from the Student Model. The ITS ELM-ART [24] includes this feature, presenting the student with a listing of all domain concepts with an associated percentage rating (a skillometer) of their understanding for each. Students are able to revise the student model with new values representing how they feel they are really progressing in a specific area. The problem is that the Open Student Model used in ELM-ART has the same structure as the courseware and therefore conveys the student's progress through the course, rather than their conceptual understanding of the course concepts.

Although domain constraints are an effective measure for evaluation of student progress [35], the grain size is too fine to effectively represent the Open Student Model. When too much information is presented, the student is able to visualise the system and as a result is unable to infer their domain understanding. The Student model is built from statistical analysis of the constraints, which are each classified under one or more domain concepts. Therefore the student can be presented with a visual model of the domain concepts, supplemented with averaged data from the Student Model. This does not impose a course structure and by applying filters, can show concepts students have learned, what they are learning, and what the relationships between these concepts are.

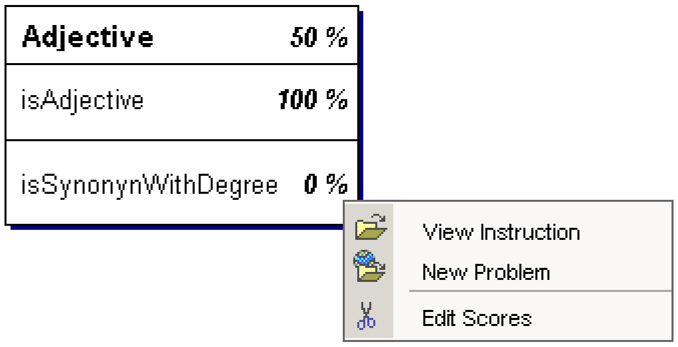

Figure 4 : Drill down on a domain concept node to show syntactic (upper) and semantic (lower) constraints. Selecting constraint gives additional options.

Through a domain concept diagram elements in Figure 4, the student is able to access instructional material or problems, that are relevant to a domain concept. The student can also use their self-assessment skills [36] to determine the accuracy of the student model and modify it if necessary. Finally, they may also drill-down to the constraint level to receive more detailed explanation of their current state of domain understanding. Buggy constraints, if present in the domain model, can provide valuable information for the student through the Open Student Model. As an example, for a mathematics tutor, three levels of feedback could be supplied when the student queries the *addition* concept:

a) You are still having difficulty with addition. Here is the lesson on addition… (Conceptual level).
b) You have not mastered carry digits for addition. Here is how you use a carry digit… (Constraint level)
c) You often forget to use a carry. You also sometimes add the carry to the wrong column. Here is what you should do in these situations… (Buggy Constraint level)

Supplying the student with too much feedback on student solutions during a problem session is actively avoided, as it does not encourage them to think for themselves. However, if the student is not in a problem solving session and are reviewing their performance, they are seeking a different type of feedback. They wish to know how well they are achieving and what their common mistakes are. This way, students benefit from having control over their own learning [24]. In many cases, students simply wish to know exactly what they need to study and therefore the more detail within the Open Student Model, the more effective it becomes.

4.3 Domain Definition

The domain definition is a new concept for ITS's; it provides a complete and formal meta-description of the domain syntax. The semantics are not included as these are captured by the domain constraints. It is at the same level of abstraction as the domain knowledge, yet it is platform-independent (where the ITS is considered the platform). The definition can therefore be used in any ITS or software system that has a use for it. For most domains the identification of the syntactic structure is routine and can be easily defined by a domain expert. The purpose of the domain definition is to provide a definition language and syntactic validation for each of the ITS documents[2]: questions, solutions, constraints and even the user interface.

Each ITS document element that pertains to the domain must be specified using only meta-data that exists within the domain definition. This requirement gives the illusion of adding a high-level of complexity to the ITS structure as the domain definition is platform-independent. Yet, through the use of XML and the MDA practices, this level of integration is relatively simple to achieve.

The domain definition is defined using XML Schema which is a core component of the MDA. XML Schema is used to define meta-languages and is a strictly typed language; there are few instances that a definition can be written in multiple ways. Those inconsistencies that do exist are easily remedied with a XSLT stylesheet, which transforms the XML Schema to a compatible form before use. This stylesheet is

[2] It is always assumed that static ITS components are managed by an authoring tool or shell.

generically applicable to all XML Schema definitions as it acts on the language rather than the resultant data model.

The power of the MDA is that all of the ITS documents are defined at both the platform-independent and platform-specific level. Generic XSLT stylesheets are provided that convert between the two levels. As an example, the pure XSL constraint from Listing 3 can be transformed into the WETAS constraint from Listing 1 using a generic stylesheet. The result is that all of the XML definitions for constraints, questions and solutions proposed throughout these sections can be provided as a higher-level language defined over the standard WETAS interface. No changes must be made to WETAS whilst most of the benefits are retained.

Despite the advantages, there is a great amount of work involved with creating these generic stylesheets as they must be capable of converting, in the case of constraints, from one complex pattern matching syntax to another. Ultimately, an ITS authoring tool must be built that uses pure XML definitions for all the ITS documents, eliminating the need for a multi-level model.

4.3.1 Design Considerations

A great benefit of using the MDA and XML Schema to model the domain definition is that many domains have already been modelled using XML Schema. Domains like the ER notation provided in Listing 8, or UML Class Diagram notation that is due for free release in XMI version 2.0 [33]. Many organisations are using XML technologies to define their business processes and technologies. Work based training systems are notoriously difficult to create, yet such systems can be derived from the internal systems already in place. This notion introduces great potential benefit for ITS's.

The modelling notation used to design the domain definition is not a confounding factor. The ITS assumes no prior knowledge of the domain content or structure; it is not aware of the difference between a well defined or badly defined domain definition. A bad design will result in questions, solutions and especially constraints, that are difficult to define, yet the system will still function. An example situation follows:

The domain model can be designed using an Entity Relationship diagram, which is a conceptual data model. Once the ER diagram is complete the Xere algorithm[3] [37] can be used to generate an XML Schema domain definition. Alternatively, an implementation data model such as the richer UML notation can be used to model the domain. In this case the derived XML Schema will be substantially different from the ER data model as they were created using different constructs. The result is that both systems have operational domain definitions. The ER model is based on a simple data model and can therefore only define simple ITS documents. The UML model is detailed yet requires more effort in the subsequent definition of the ITS documents.

The domain definition does pose one constraint on the user interface component. The user interfaces must confine the student to creating solutions that use the domain definition constructs. If the student generates a solution containing undefined meta-

[3] The Xere Algorithm aims to create interoperability between XML schema and ER diagrams. Its process of taking an XML schema, transforming with a XSL stylesheet and generating an XML ER diagram, is the same as the domain learning scheme presented in Section 5.3.

data the system will not be able to process it. This error will not be captured as the domain definition is never used for direct validation. Due to the nature of XML Schema, a document that does not conform can not be loaded, even if it is well-formed. As a result the student would never be able to enter a solution with invalid syntax and would result in an unusable interface.

4.3.2 Namespaces

The MDA allows the integration of multiple domain models. When domains become large and distributed over different sub-domains, the internal representation of elements have a risk of clashing. Programming languages and modelling notations are an example; the ER diagram notation for attribute is different from the UML Class Diagram attribute notation, which in turn is different from an attribute in Java. A tutor that incorporates these domains into a centralised ITS domain model would not operate correctly. Manually merging the two domain definitions is infeasible and unnecessary. XML namespaces [38] manage this problem by qualifying domains using Uniform Resource Identifiers (URI). The probability that there exist two domains with the same name and conflicting URI's is extremely low. The XML Schema defines the constraints of the namespace; each document that uses it must consequently be a part of that namespace.

4.3.3 XML Schema Definition

The following listing shows a sample of the ER domain definition, the full listing is in Listing 8. This defines an *Entity* element that consists of zero to many *Attribute* elements, where order is irrelevant, and an attribute *name* that is required and must be unique with respect to all other *Entity* elements.

```
<xs:element name="entity">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="attribute" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:key name="entity_key">
      <xs:selector xpath="."/>
      <xs:field xpath="name"/>
    </xs:key>
  </xs:element>
```

Listing 1: An XML Schema definition for an *Entity* with the ER domain.

4.4 Domain Knowledge

Constraints capture the declarative syntax and semantics of the domain. These are the relationships and interactions between elements of the domain definition that can not be defined using XML Schema, or that are subjective as they depend on a certain state of the system to be valid (Section 4.4.1). Constraints represent the domain facts that group together to form the domain concepts; constraints are categorised and associated with the domain concept model (Section 4.2). Constraints perform pattern matches on the question and solution XML documents that conform to the XML Schema domain definition. Constraints are therefore defined using XSLT stylesheet, which takes one XML document, performs a transformation and generates a new textual document. XSLT can only perform transformations on a well-formed XML document, although any output format may be used. A stylesheet defines templates that search XML documents, using XPATH queries, for specific meta-tags. Once a meta-tag has been located it outputs some feedback into the new XML document. The template illustrated in Listing 3 locates *Entities* with duplicate *names* and generates an XML formatted feedback message.

Domain knowledge in a CBM tutor is represented using constraints, in a MT tutor it is represented using production rules and a bug library. Therefore, the domain knowledge is platform-specific as it is dependent on the type of ITS it is created for. With consideration to the MDA there are two platforms within an ITS; the type of ITS and the language used to write its various facets. Similar to the domain definition the constraints are defined at each of these levels. The ITS platform constraints are defined using XSLT (Listing 1) and are therefore platform-independent for that specific platform. This means that CBM constraints can be integrated with any other CBM ITS. Constraints at the language level are generated from the XML constraints; and can therefore be defined using any syntax.

Another advantage of using XSLT is that it allows constraints to reference each other, rather than rely on external macros or functions [16]. This allows for reuse within the constraint set, reducing the redundancy. The relevance or satisfaction condition of a given constraint may encapsulate that of another constraint, this is common in a hierarchically structured domain where parent constraints abstract their child constraints [39]. A constraint must be able to reference the relevance and/or satisfaction condition of another constraint. This binds constraints together which is important to ensure the integrity of the domain model and it simplifies the automated generation of constraints, for which the generation of macros would be problematic.

4.4.1 Modal Constraints

Modal constraints are an addition suggested for domain modelling. Syntactic and semantic constraints are able to capture facts of domain concepts, and through the relevance condition the system is able to determine when it is appropriate to use them. With open-ended domains there are multiple solutions to a given problem. In this case the constraints are engineered to check dependencies that generate alternative solutions.

This works well for small domains where the constraints are independent, yet in larger domains there are different styles of solving problems. Concepts can overlap and are applied differently. When two overlapping concepts are validated concurrently they conflict and inappropriate feedback is generated. Constraints are unable to capture this as they are not defined at the conceptual level. In this situation there are two groups of constraints that conflict if processed together but will produce the correct feedback if only one group is evaluated. An example is in the UML notation domain, where the notation remains constant, yet there are many modelling techniques that are each considered correct. Another example within the same domain is when advanced students are expected to use Object-Oriented design patterns, yet beginner students are not.

The solution is to make constraints modal. This can be achieved by allowing the constraints relevance condition to check against external variables. These variables are generated by the student model or specified by the student. As illustrated in Figure 5 the student would be able to select a modelling technique to attempt.

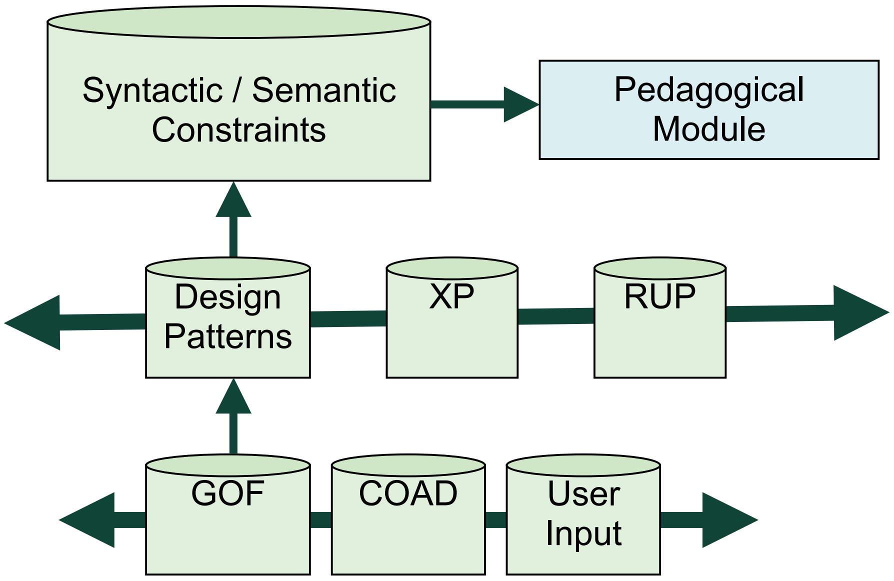

Figure 5 : Illustrates how modal constraint repositories can be shuffled and selected.

4.4.2 Feedback

The feedback from an ITS is controlled by the pedagogical module, which implements the teaching strategies [14]. Yet, constraints have their own feedback message hard-coded, decreasing the adaptability of the ITS. In an ITS with a hierarchical domain it is logical to provide feedback messages that mimic this structure. It can be achieved in two ways, firstly by providing constraints at each level of the hierarchy, each with a feedback that generalises the child node. Secondly, by using a meta-language for the definition of feedback messages, an algorithm can generate abstracted, natural language feedback messages that contain references to all violated child constraints.

Another advantage of this approach is that unique feedback messages can be created. Adaptability and uniqueness are important concepts for Intelligent Tutoring. By using a meta-definition for feedback messages they can be generated uniquely. As an example, the feedback message "*Check the names of entity types. They must be unique*", could be rearranged to produce "*Ensure that each of your entity types have unique names*", or "*Unique names must be supplied for all entity types*". This is one case where the fickleness of English is useful, as each message is sufficiently different that it fools the mind to thinking that they convey a different meaning. If a student is consistently presented with the same feedback message they run the risk of learning how to fix an error based on the message, rather than on the root cause of the problem (as happens with compiler errors). Using dynamic feedback messages, unless the student is deliberately searching for short-cuts, they will be forced to think about all feedback messages.

This could be implemented in two simple ways. The first method has a list of common types of feedback message that relate to groups of constraints. For each type a list of alternatives is defined, where key words can be substituted from the feedback message definition contained within the constraint. This is not optimal as there would be a limited number of combinations possible for each message. The second method is to create a list of similes and different word arrangements to common phrases. The feedback messages are defined with meta-tags that refer to these phrases. The pedagogical module is responsible for selecting an appropriate simile for each substitutable meta-tag. By changing both words and their arrangement, more unique message can be generated.

4.4.3 XSLT Definition

```
(10
  "Check the names of your entity types. They must be unique."
  (match SS ENTITIES (?* "@" ?tag1 ?ent_name ?*))
  (not-p
    (match SS ENTITIES (?* "@" ?tag2 ?ent_name ?*
"@" ?tag3 ?ent_name ?*))
  )
  "entity types"
)
```

Listing 2 : A WETAS constraint that checks all *Entity*'s have unique *name*s for the ER domain.

```
<!-- Relevance Condition -->
<xsl:template match="Entity">
  <!-- Satisfaction Condition -->
  <xsl:if test="name=preceding::Entity/Name">
    <!-- Constraint output -->
    <xsl:element name="ConstraintID">10</xsl:element>
    <xsl:element name="OntologyMember">
      Entity
    </xsl:element>
    <xsl:element name="Feedback">
      Check the names of your entity types. They must be unique.
    </xsl:element>
  </xsl:if>
</xsl:template>
```

Listing 3 : An XSLT template constraint that checks all *Entity*'s have unique *name*s for the ER domain.

4.5 Courseware (Problems and Solutions)

The courseware is not a part of the domain model, yet it is strongly influenced by it. The courseware must be defined in the same language that is used to interpret it; in this case the courseware is defined using the domain definition. In WETAS questions and solutions are defined using LISP syntax. People entering courseware are not necessarily familiar with computers and require a simple interface [18]. To make this more effective the courseware must be represented using XML. It allows for integration with the domain model notation and also to provide a simple means of building a visual courseware entry mechanism for ITS shells. As the questions and solutions are specified using the domain definition they provide validation to ensure accuracy.

A major advantage of representing the question and solution using the domain definition is that the system can easily determine the relevance of problems to specific constraints. This aids problem selection and makes the tutor intelligent without hard-coding these relevance relationships. The system enumerates each of the elements that exist within both the question and solution, and are represented in the domain definition. These elements must belong to at least one constraint relevance or satisfaction condition. Those constraints can be judged as to their level of relevance through the complexity of the constraint and how many domain elements were used from the question and solution. This process can also be performed in reverse to determine the relevance of a constraint to a problem.

The representation of questions internally using pure natural language is a disadvantage as it is difficult to extract information without any direct relationship with the domain definition. In KERMIT [6] questions are supplemented with meta-tags that are interpreted by the constraints. These meta-tags have no direct relationship to the domain, and therefore an extra layer of knowledge must be added to the tutor to understand them.

A more elegant solution is to define the question using the domain schema definition. Constraints can then refer to question content as they would the ideal or student solution, as both have the domain definition as the common language. An algorithm similar to that explained in Section 4.4.2 is used to convert the meta-definition to natural language. This further leads to question generation opportunities.

4.5.1 Question Generation

Using the domain definition creates the opportunity for automated generation of the question and solutions. In [40] a technique is proposed for automated generation of courseware from constraints. Constraints represent fragments of the domain. By selecting a set of constraints a unique question and ideal solution can be extrapolated. This is a complex procedure and may not be effective for all domains. An advantage of using the domain definition is that the appropriate representation for the question and ideal solution is given. They must simply select a set of entities and attributes from the definition and define a set of relationships between them. This selection may be random or based on the relevance condition of a set of target constraints generated by the student model. This allows the system to generate problems that are relevant to the specific students. Once the template has been generated, a problem domain (e.g. school, transportation, movies, etc) is overlaid to give the student some context.

4.5.2 Validation

Errors within the question and solution sets are inevitable. The ITS can provide a function to help in locating them. If the ITS collates each of the students solutions, either verbatim or in a global student model, it can determine patterns in the students answering behaviour. As an example, if a majority of students answer a particular question correctly yet in a method different from that suggested by the ideal solution it suggests one of two things; either the students were not adequately taught that concept or that the ideal solution is inappropriate. While the ITS will continue to operate correctly with an inappropriate ideal solution, if that solution is presented to the student it may cause confusion.

4.5.3 XML Definition

```
(10	; problem number
	10	; difficulty
	"10. 10. Sometimes <E1> students </E1> <R1> work in </R1> <E2> groups </E2>. Each <E2> group </E2> has a unique <E2K1> number </E2K1> and <E1> students </E1> have their <E1K1> student ids </E1K1>. A <E1> student </E1> may have different <R1S1> roles </R1S1> in various <E2> groups </E2> he/she belongs to."
	(("ENTITIES" "@ E1 STUDENT regular @ E2 GROUP regular")
	 ("RELATIONSHIPS" "@ R1 WORKS_IN regular")
	 ("ATTRIBUTES" "@ E1K1 Id key simple E1 @ E2K1 Number key simple E2 @ R1S1 Role simple composite R1 ")
	 ("CONNECTIONS" "@ C1 partial N R1 E1 @ C2 total N R1 E2 "))
	"10.gif"
	"Student groups"
	)
```

Listing 4 : A WETAS style question and ideal solution.

```
<Problem>
  <Number>10</Number>
  <Difficulty>1</Difficulty>
  <Description>...</Description>
  <IdealSolution>...</IdealSolution>
  <!--Optional/Customised -->
  <Schema>Student</Schema>
  <SolutionImage></SolutionImage>
</Problem>
```

Listing 5 : An XML question definition.

```
<ER>
  <Entity>
    <Name>AIRPLANE</Name>
    <Attribute>
      <Name>Business_Seat</Name>
      <Type>Simple</Type>
    </Attribute>
    <Attribute>
      <Name>Economy_Seat</Name>
      <Type>Simple</Type>
    </Attribute>
    <Attribute>
      <Name>Year</Name>
      <Type>Simple</Type>
    </Attribute>
    <Attribute>
      <Name>Reg_Number</Name>
      <Type>Key</Type>
    </Attribute>
    <Attribute>
      <Name>Type_Attribute</Name>
      <Type>Simple</Type>
    </Attribute>
  </Entity>
</ER>
```

Listing 6 : An XML definition of the ideal solution from the question defined in Listing 4.

5 Knowledge Acquisition

This section explores a range of techniques that might be used to simplify or automate the creation (or learning) of the domain knowledge.

Once the domain modelling technique has been selected (in this case CBM) the domain model must be built. The first step is to create the Ontology; the domain definition is a formal definition of the domain syntax, the domain concept model is a high-level representation of the domain concepts. The domain definition may be available from another application that represents its information using XML Schema. Otherwise, the domain definition must be constructed by a domain expert. The accuracy of the domain model is critical, if parts of the domain are incomplete the ITS will be unable to process effectively. All of the knowledge acquisition techniques rely on the domain definition. The domain concept model is less critical depending on how the underlying ITS Authoring Tool or shell implements it. This is used to categorise the domain constraints and it used for some of the knowledge acquisition techniques covered in this section.

For CBM tutor knowledge acquisition, the process involves eliciting the platform-specific constraint set. These constraints are extracted from the declarative syntactic and semantic dependencies found within the domain definition. The MT domain modelling approach is not the focus of this research; it captures the procedural domain concepts in the form of production rules and a bug library. This bug library is problematic for automated acquisition as it contains all the invalid actions a student may perform. Knowledge discovery algorithms require input that proves examples of student behaviour or domain concepts. For any significant domain, the number of negative examples (or bugs) present is combinatorial, making this approach prohibitive.

The most common approach toward determining the domain knowledge is best described as *manual* [14]. Automated knowledge acquisition is a much touted goal for ITS's [14, 18, 20, 41], yet there are few practical implementations {Murray, 1999 #26}. The best ITS authoring tools are able to greatly aid the input procedure, yet it still remains manual. The problem of knowledge acquisition is a complex one and is typically classified under machine learning. The difficulty with machine learning tools is that they are aimed toward general knowledge acquisition with arbitrary domains and applications. In the following sections the nature of the ITS domain knowledge and domain definition is exploited to demonstrate practical techniques for the implementation of automated domain constraint acquisition tools.

5.1 Primer

One generally applicable simplification for the constraint acquisition process concerns the verification of constraints. This can be achieved using the same technique that is used for the validation of questions and solutions (Section 4.5.2). There is a fortunate property of constraint-based domain models in that if it is incomplete it can still be used, yet it will generate false positive solutions during validation. Therefore, as the ITS is built and new constraints are added, questions and solutions are presented to test the accuracy of the model. The system records all solutions presented to it and the state of each constraint after the solutions are validated. When a new constraint is entered into the system, every solution relevant (i.e. they share domain definition elements in either the relevance or satisfaction condition) to the new constraint is revalidated. Constraints are independent, so adding a new constraint will not affect the result of any old constraint's validation. However, this new constraint should cause new failed solutions and/or old failed solutions with additional feedback from the new constraint.

The authoring tool should visually display these result to the domain expert. Additional statistics can be generated that determine the effect across the system. For example a 100 percent effect would indicate that the new constraint contains a tautology, whereas a zero percent effect indicates that the constraint is never relevant. An ITS authoring tool must assume that the domain expert is not an ITS or computer expert and that log based debugging is inappropriate. Once the domain expert is satisfied with the new constraint, the model is updated with the new validation state of each question and solution.

5.2 Programming by Example

In B. Martin's paper [42] he describes a method of teaching the CBM ITS domain constraints by having a dialog between the domain expert and the system. The system generalises terms from the problem and solution to test some underlying domain concept. For each new concept an example is supplied, using its knowledge the example is generalised and presented to the domain expert. If the example is rejected, the new concept is false. If it is accepted, the system attempts to further generalise until the generalised concept is rejected, at which point the constraint is generated. The technique is based on MARVIN machine learning system [43] from 1986.

The following technique is an adaptation of B. Martin's "Learning by asking Questions" approach. The ITS attempts to mimic the relationship between tutor and student; the computer is now the student and the domain expert is the tutor. The domain expert supplies questions and solutions, and the system searches for relationships and patterns that can be used to form constraints. The ITS is initialised with the domain definition that gives it the ability to process domain inputs such as questions, solutions and constraints. Based on its current constraint set and a set pre-programmed heuristics, the system is able to make alterations to questions and solutions. This forms a new example which is validated by the domain expert. From the domain expert's answer, new constraints can be inferred from the changes that were made and their effect on the solution. This reduces the machine learning problem to the permutation of well-defined XML document elements and pattern matching.

5.2.1 The Algorithm

This approach has two phases. The first phase aids the domain expert in locating and creating the core domain constraints. It requires that the domain definition has been created and that the domain expert is able to create questions and solutions using that syntax, and that they are able to critique solutions. This technique works equally well for finding positive or negative examples of domain knowledge. In fact, it must store buggy constraints otherwise the algorithm will never exhaust the list of possible permutations that might be made to the domain inputs.

The system has a list of heuristics which determine how XML Schema elements can be permuted. These heuristics are ordered by their level of severity (how drastically they alter the input). As an example, the removal of an XML *Entity* or *Attribute* is considered severe, whereas changing the case of an XML *Attribute* is considered minor. Possible permutations include: changes to text, removal of an element, addition of an element, combination of elements and changing an element type. The order that the permutations are applied has a significant effect on the operation of the algorithm. Practical implementation is the only way to determine the most appropriate order.

Step 1	The domain expert supplies the tutor with a question and ideal solution using the representation from Section 4.5.3.
Step 2	From the domain definition the system enumerates all of the recognisable elements within the question and solution. The grain size is determined by the level of detail within the domain definition.
Step 3	Based on the current domain knowledge, the domain definition elements are tagged if they have one or more associated constraint.
Step 4	For those elements that are not tagged, their state is altered using a heuristic, either structurally or through its text. This produces a new solution.
Step 4.1	If there are no permutations of the inputs possible, the algorithm is complete. This occurs when constraints prohibit any possible alterations to the solution. For this reason buggy constraints must be included.
Step 4.2	The newly generated solution is presented to the domain expert.
Step 5	The domain expert validates the new solution, corrects any errors and resubmits the solution.
Step 5.1	If the resubmitted solution is the same as Step 1, the domain expert has changed the answer back. A constraint is generated.
Step 5.2	If the resubmitted solution is the same as Step 4, the adjustment did not affect the syntax or semantics of the solution. The algorithm returns to Step 4.
Step 5.3	If the resubmitted solution is new, the system repeats step 4 until it is able to generate a new solution, or asks the domain expert for the sequence of alterations. A constraint is generated.
Step 6	Repeat from Step 1.

When a new constraint is created the system presents the domain expert with both a template constraint that encapsulates the alterations, and an ordered list of those alterations. From this, the domain expert is able to interpret an accurate constraint definition. The new constraint is subsequently added to the domain knowledge and categorised in the domain concept model.

The benefit of this approach over programming by demonstration is that the expert is not required to think of all the different ways to correctly and incorrectly solve a problem.

5.2.2 Programming by Demonstration

Programming by Demonstration was used in Demonstr8 [20] (Section 3.1). For this technique the system is supplied with examples from the domain expert that illustrates how a real student would solve a problem. The system attempts to infer what process is taken and subsequently generate domain knowledge. In many cases the domain modeller is an educator and has access to questions and real correct and incorrect student solutions. In the programming by example technique all that is required is the ideal solution and the system attempts to discover what changes could be made. By demonstrating to the system typical student solutions, differences between these and the ideal solution can be identified. The difficulty is that identification of these differences and what was the cause of them is difficult. The technique of applying small and cumulative alterations to the student or ideal solution can be used to determine what has changed.

This introduces a problem as a student does not necessarily violate one constraint. The system has no way to differentiate between the different solution paths, except by the rigorous comparison of all student solution paths for that problem. The system first locates all solutions that have no errors and generates constraints for those that have changed elements. It then attempts to locate solutions with few changes but which are incorrect. Constraints are generated that capture these changes and as the constraint set is becomes more complete, more complex constraints can be derived.

Although in theory this process is fully automated, the level of processing required to perform the operations required would be immense. The location of a path between ideal solution and student solution is very difficult and has been the subject of much research, as it is the underlying concept of Model Tracing tutors. Programming by example is likely to be the more effective approach.

5.3 Transformation Approach

The transformation approach is fully automated and only relies on the domain expert to provide the domain definition. The hard work is performed once initially by the ITS expert when creating the ITS Authoring tool expert, who must create a set of XSLT templates that are capable of converting the domain definition elements into constraints. Analysis of existing constraint sets for KERMIT and SQL-Tutor, show that there is a strong relationship between the constraint conditions and the structural composition of the domain.

Based on analysis of pre-existing domain constraints a set of transforms are created that convert domain definition patterns to common constraint type. Listing 7 shows a XSLT template that search the domain definition for an *xs:Element* with an *xs:key* element (i.e. all elements with unique attributes will be located). It proceeds to extract the name of the entity and the attribute. Finally a template constraint is generated, complete with unique identification number and generic feedback message. The reference *ConceptMember* links the constraint to the concept domain model. This acts simply as a reminder as this technique is not able to extract the concept domain model as the abstract domain concepts are not represented within the domain definition. This is a very simple example constraint transform; complex constraints defined using XSLT transforms quickly become very long and hard to define. Transforms can only be defined by highly skilled use of XPATH queries.

The ratio of XSLT template to generated constraint is not one-to-one. This transform, when applied to the ER domain definition, creates two constraints. In some domains, such as LBITS, the constraints are very similar where a few templates can generate a large proportion of the domain model. The important result is that this template is generic and can be applied to any other domain definition. Subsequent constraints are generated for free.

```
<xsl:template match="//xs:Element">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="xs:Key/xs:Field">
  <xsl:variable name="EntityName"
select="ancestor::xs:Element/@name">
  <xsl:variable name="AttributeName" select="@xpath">

  <!-- The following is pseudo xslt output -->
  <xsl:template match="$EntityName">
    <xsl:if test="name=preceding::$EntityName/$AttributeName>
      <xsl:element name="ConstraintID">
        <xsl:number/>
      </xsl:element>
      <xsl:element name="ConceptMember">
        $EntityName
      </xsl:element>
      <xsl:element name="Feedback">
        An attribute [$AttributeName] from entity [$EntityName] must
be unique.
      </xsl:element>
    </xsl:if>
  </xsl:template>

</xsl:template>
```

Listing 7 : Example of a transform between the ER domain definition (Listing 8) and the constraint defined in Listing 3.

5.3.1 Analysis

A limited version of the ER domain definition is provided in Listing 8. This domain definition is captured by 7 syntax constraints and 24 semantic constraints.[4] These constraints were taken from KERMIT. Of these constraints, 3 syntax and 20 semantic constraints can be generated directly from the domain definition by XSLT transformations. The domain definition shows that the names can not be shared between Entities and Relationships. It shows the incorrect cardinality for entity

[4] The constraints used in this evaluation were taken from the KERMIT ITS.

attributes and it does not define the cardinalities values for relationships. Of these only the cardinality values can not be defined using XML Schema. The final result is that, of the 31 constraints, 27 can be generated with an appropriate XSLT stylesheet that captures these conditions. It must be noted that the ER domain definition supplied is very simple; the full ER domain model consists of 26 syntax and 66 semantic constraints. Many of these constraints have more complex conditions.

5.3.2 Limitations

The greatest limitation of this approach is that it only works for domains that are structured similarly to XML Schema. For example, a superlative adjective within a language tutor might be defined as ending with '*est*'. This can be defined in XML using a *simple type* and a *pattern restriction*, yet this definition is specific to that domain fact. It is unlikely that a defining an XSLT transform for this constraint will ever be used by another domain. This problem will decrease as transforms are created for a wide range of domains. However, domains that contain many unique definitions are likely to always be a problem. Fortunately, there are a large number of domains that can be defined using the general XML Schema constructs, including programming languages and software design notations (including ER and UML).

Another limitation is that resulting constraints are static with respect to the domain; they are unable to infer alternate solutions. To an extent, the generated constraints can act simply as a guideline and as a means of locating trivial constraints. A legitimate advantage is that these constraints can be used to initially populate the domain for subsequent use in other domain learning techniques that require them.

5.4 Inquiry-Based Approach

This high-level approach is aimed at aiding the domain modeller to create the whole domain model. It does not automate the generation of constraints as with the previous approaches. In previous techniques it was assumed that the domain definition was predefined or easily created by the domain expert. Inquiry-based machine learning algorithms have been a popular technique for knowledge acquisition [44]. The concept is that the domain expert provides domain input, the system then asks questions to infer semantic knowledge of that domain.

Specifically, the system has a repository of pre-defined questions or question types that explore an aspect of the domain definition. The inquiry facility offers no automated acquisition of domain model components; it serves to aid the domain expert's mental exploration of the domain. Therefore, the following two options explore concurrent and consecutive methods of integrating inquiry-based learning with the Transformation approach explained in Section 5.3.

The first option explores consecutive integration where the Transformation approach is run to completion and subsequently the inquiry phase begins. Once all of the constraints have been generated the system asks questions to ensure that the domain expert has critiqued the domain knowledge. These questions can take two forms; direct generalisations of the constraint transforms (each transform is pre-programmed with different scenarios) and questions that explore the relationships. Relationships are between the classification of constraints within the domain concept model and constraints relevant to the domain definition. As the transformation approach is unable to capture all domain constraints, especially those complex ones, the inquiry system aids the developer in completing the model. This only works in situations where the domain model is pre-built.

The second approach integrates the inquiry facility with the creation of the whole domain model, no prior specification is assumed. Domain constraints are generated (through transforms) while the designer creates the domain Ontology; the domain concept model and domain definition. Concurrently, the system generates questions that probe the domain modeller's decisions and actions to infer semantic knowledge. While the constraint templates are necessarily scoped to the domain definition, the questions are able to explore the linkages between the domain definition, concepts and constraints. When the domain modeller reaches an impasse, the system is prompted to suggest aspects that have not been considered. These questions relate to elements or structures of the domain definition or concept model that have few linkages. As explained in Section 4.4.2 it is important that these questions are generated uniquely to support a creative rather than a robotic approach.

As a trivial example, when designing a music tutor, the domain expert creates an entity for *Note*, a number of *Note subclasses* and an Entity for *Bar*. The *Bar* has an aggregation of *Notes*. The system now has enough information to ask whether the order of elements for the aggregation is important. In this case it is, so the system generates a generic constraint that is capable of checking sequential aggregations. The expert amends this with domain knowledge of how exactly the sequence is important. At this stage the designer realises that the definition is too shallow. In the Music domain the sequence of notes is of critical importance and has many constraints governing it. Consequently, the domain expert discovers that note sequencing is a fundamental concept of the Music domain and it is added to the domain concept model.

As designing a domain Ontology is not a linear process, there are likely to be many revisions before a final solution is developed. The generation of constraint templates and questions act as promptings to aid in the full exploration of the domain concept currently being considered. Despite the volatility of the domain Ontology during the design process, as constraints are considered to be the lowest level form of domain knowledge, they will require little or no revision after they are first derived.

6 Future Work

Many new concepts have been described throughout Section 4 and 5 for the modification or extension of existing ITS processes. This section describes future applications of these suggestions and introduces some other areas that lead from the findings of this report.

6.1 Build a MDA ITS

Section 4.1 describes a new approach to domain modelling by splitting the domain model into three platform-independent components: domain concept model, domain definition and domain knowledge. Each of these is physically represented using XML and related technologies. These components are overlaid on an ITS using the MDA. The next logical step is to implement and evaluate these techniques. Initial experiments can be incorporated into an existing ITS Authoring Tool such as WETAS. This would be sufficient to prove the effectiveness of the MDA and to develop standards for the representations. If successful this would lead to the development of a pure XML ITS.

XMLTutor [45] is an ITS defined using XML, to teach XML. It encapsulates many of the ideals presented in this research but it does not use the MDA. In this pure XML ITS only MDA technologies would be used for the implementation, this includes Java – for components, XML – for documents and data interchange, and SOAP [46] – for distributed internet-based communication. This excludes LISP and therefore most of the framework of WETAS. Fortunately, XML is well supported on all major operating systems and low-level frameworks already exist for processing it. The selected platform must support XML pipelines. These are critical to the simple implementation of the MDA for an ITS Authoring Tool. An XML pipeline takes an XML document and applies a series of XSL transformations to generate some final output. An example of this process is shown in Figure 6. The pipeline allows for the implementation of a modular, component based architecture. One powerful Java-based platform that supports this notion very well is Cocoon[5], which is aimed at creating XML/Java, component-based websites.

The suggested approach is to build an ITS that encapsulates the domains of KERMIT [6], NORMIT [47] and SQL-Tutor [23]. These are all tutors within the greater domain of databases and each represent different problem styles. As mentioned, the underlying platform could be WETAS, or implemented using an XML based platform such as Cocoon. This is the power of the MDA. The process would require the creation or locating of XML Schema definitions for each domain. The constraint and question set must then be converted to the XSLT and XML representations. This is not a small task, however, it is menial as the constraint set has already been proved to work. Once this is complete the interfaces for three sub-domains must be created. This must communicate using XML to the platform-independent domain model. Depending on the platform-specific architecture, messages are either processed directly, or have a stylesheet applied that converts them to the WETAS format for processing. This is only a very brief description of the system. The important observation will be how effectively constraints can be represented using XSL, how

[5] More information on Cocoon can be found at: http://cocoon.apache.org/

well the three database sub-domains integrate and how efficiently the MDA can be implemented.

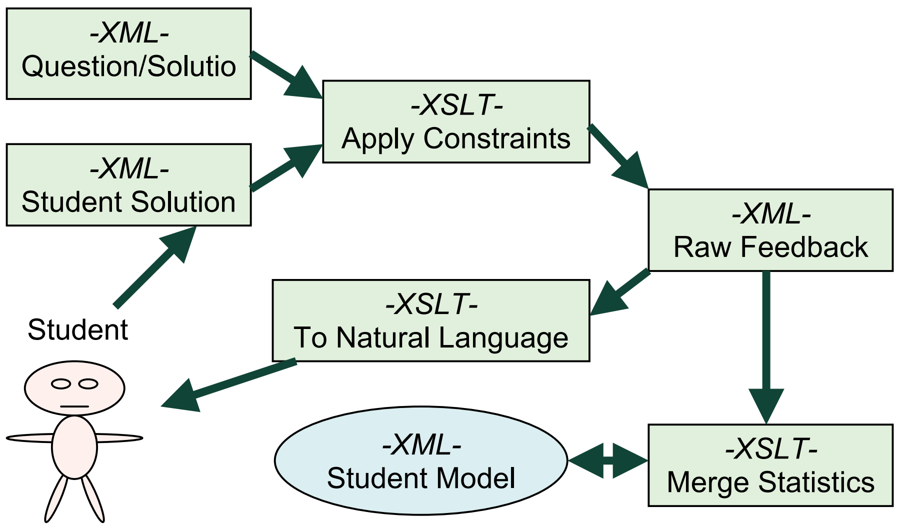

Figure 6 : Demonstrates an XSLT pipeline that takes a student solution, generates feedback messages and updates the Student Model.

6.2 Evaluate Transformation Approach

A practical evaluation should be performed on the Transformation approach to determine how effective it is when applied to a real domain. This may follow from the creation of the MDA ITS described in the previous section, as it has many of the same requirements. It is suggested that a new tutor for UML Class Diagrams be built, based on transforms generated from the KERMIT, SQL-Tutor and NORMIT constraint sets. UML is an ideal choice as the domain shares similar concepts, syntax and semantics, to these existing ITS's. As UML is a rich notation it is likely to include constraints from each of these domains.

The Process:

1. Identify patterns within existing constraint sets – ER, SQL and NORMIT.
2. Write specific XSL transformations between XML Schema.
3. Generalise transforms between tutors.
4. Apply transforms to new domain – UML.
5. Evaluate new constraints.

The first important observation will be what the level of overlap is between the transforms generated for each domain. The transforms can be specified using the WETAS constraint language, and is therefore not dependent on a MDA to support it. The appropriateness of the generated constraint set can be evaluated without a full UML ITS, however to test for completeness the UML domain model would have to be completed.

6.3 The Semantic Web

A path that was not followed is how definition of the domain model would benefit from the Resource Description Framework (RDF) [48] and the Web Ontology

Language (OWL) [49] technologies. RDF is an assertion language used for describing and interchanging metadata. It is a low-level datamodel for defining objects and the relationships between them. OWL is a revision of the DAML+OIL [50] web ontology language. It extends the vocabulary offered by RDF, defining classes of objects complete with property types, relationships and cardinalities. It was observed in Section 4.3.1 that XML Schema was unable to sufficiently define cardinalities. Therefore, RDF and OIL could be used to replace the XML Schema used to define the domain definition.

RDF and OWL are underlying technologies for the creation of the Semantic Web [51], which aims at creating an XML-based global data representation for the World Wide Web. The semantic web would be of particular interest to an XML-based ITS such as that proposed in Section 6.1, as it can serve as the domain model. The semantic web is a domain model that has the potential to eventually define representations for all forms of knowledge. If an ITS is able to integrate with the Semantic Web the effort in creating new ITS's could be reduced to simply providing instructional material and problems. This is a lofty goal, which provides an excellent target for ITS's to strive for.

7 Summary

The goal of this research was to illustrate methods that can be used to extend and simplify knowledge representation and acquisition for the creation of ITS's and ITS Authoring Tools. This research presents three significant contributions to knowledge representation and acquisition for CBM ITS's:

➢ It was shown that the domain model can be defined in three forms: the domain concepts, the domain definition and the domain knowledge. The advantage of the domain definition is that it provides a domain independent schema that unifies all of the domain model and courseware components. Furthermore, the richer domain model explicitly categorises constraints to a domain concept model. Together these extensions aim to simplify the implementation of the feature set within an ITS Authoring Tool and enable the automated generation of the domain knowledge.

➢ It was shown how the coupling can be reduced between the domain model and the ITS components by using the MDA. An XML representation for the domain model and courseware has been shown to be independent of the ITS implementation platform. The benefits are in providing the ability to integrate knowledge base of ITS's on independent implementation platforms without change to the underlying system. It also increases the efficiency of the domain model's portability, extendibility, reusability and distributed architecture. It indicates that it will possible to create standards for knowledge representation in ITS's.

➢ Finally methods of simplifying the domain knowledge acquisition were presented. The most significant of these is the Transformation approach, which searches for known patterns with the domain definition and extracts them as constraints. This has the potential to provide fully automated constraint generation for a large proportion of the domain knowledge for selected domains.

In conclusion, it has been shown that the field of Intelligent Tutoring has much to gain from recent advances in Software and Knowledge Engineering. This research provides a step in the right direction toward the ultimate goal of ITS representation standards through platform-independent domain models. The next step will be in creating an ITS that implements these solutions.

8 References

[1] A. Mitrovic, K. R. Koedinger, and B. Martin, "A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modelling," presented at Ninth International Conference on User Modeling (UM 2003), 2003.

[2] J. E. Beck, B. P. Woolf, and C. R. Beal, "ADVISOR: A machine learning architecture for intelligent tutor construction (2000)," presented at Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, Austin, Texas, USA, 2000.

[3] C. P. Bloom, "Promoting the transfer of advanced training technologies," presented at Third International Conference on Intelligent Tutoring Systems, Montreal, Canada, 1996.

[4] E. El-Sheikh and J. Sticklen, "Generating Intelligent Tutoring Systems from Reusable Components and Knowledge-Based Systems," presented at 6th International Conference in Intelligent Tutoring Systems, Spain, 2002.

[5] A. Mitrovic, M. Mayo, P. Suraweera, and B. Martin, "Constraint-Based Tutors: A Success Story," presented at Fourteenth International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems, Budapest, Hungary, 2001.

[6] P. Suraweera, "KERMIT: A Knowledge-based Entity Relationship Modelling Intelligent Tutor," presented at New Zealand Computer Science Research Students' Conference (NZCSRSC), New Zealand, 2001.

[7] S. Ohlsson, "Constraint-Based Student Modeling," *Artificial Intelligence in Education*, vol. 3, pp. 429-447, 1993.

[8] A. Mitrovic and B. Martin, "Evaluating Effectiveness of Feedback in SQL-Tutor," presented at IWALT 2000, Palmerston North, 2000.

[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)," World Wide Web Consortium, Technical Report REC-xml-20001006, 6 October 2000.

[10] D. C. Fallside, "XML Schema Part 0: Primer," IBM, Technical Report REC-xmlschema-0-20010502, 2 May 2001.

[11] J. Clark, "XSL Transformations (XSLT) Version 1.0," Technical Report 16 November 1999.

[12] S. D. James Clark, "XML Path Language (XPath) Version 1.0," Technical Report REC-xpath-19991116, 16 November 1999.

[13] J. D. Poole, "Model-Driven Architecture: Vision, Standards And Emerging Technologies," presented at Workshop on Metamodeling and Adaptive Object Models (ECOOP 2001), 2001.

[14] T. Murray, "Authoring Intelligent Tutoring Systems: An analysis of the state of the art," *Artificial Intelligence in Education*, vol. 10, pp. 98-129, 1999.

[15] A. Mitrovic and B. Martin, "WETAS: A Web-Based Authoring System for Constraint-Based ITS," presented at Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Malaga, 2002.

[16] B. Martin, "Intelligent Tutoring Systems: The practical implementation of constraint-based modelling," in *Computer Science*. Christchurch: Canterbury University, 2003, pp. 218.

[17] B. Martin and A. Mitrovic, "Authoring Web-Based Tutoring Systems with WETAS," presented at ICCE 2002, Auckland, New Zealand, 2002.

[18] N. Major, S. Ainsworth, and D. Wood, "REDEEM: Exploiting symbiosis between psychology and authoring environments," *Artificial Intelligence in Education*, vol. 8, pp. 317-340, 1997.

[19] S. Ainsworth and S. Grimshaw, "Are ITSs Created with the REDEEM Authoring Tool More Effective than "Dumb" Couresware?," presented at International Conference on Intelligent Tutoring Systems, 2002.

[20] S. B. Blessing, "A programming by demonstration authoring tool for Model Tracing tutors," *Artificial Intelligence in Education.*, vol. 8, pp. 233-261, 1997.

[21] J. R. Anderson, M. Matessa, and S. Douglass, "The ACT-R theory and visual attention," presented at Seventeenth Annual Conference of the Cognitive Science Society, Hillsdale, NJ, 1995.

[22] K. R. Koedinger, V. A. W. M. M. Aleven, and N. T. Heffernan, "Cognitive Tutor Tools for Advanced Instructional Strategies," Office of Naval Research April 1st - Sept 30th 2002.

[23] A. Mitrovic, "A Knowledge-Based Teaching System for SQL," presented at ED-MEDIA/ED-TELECOM'98, Freiburg, 1998.

[24] G. Weber and P. Brusilovsky, "ELM-ART: An adaptive versatile system for web-based instruction," *International Journal of Artificial Intelligence in Education*, vol. 12, pp. 351-384, 2001.

[25] A. Mitrovic and V. Devedzic, "A Model of multitutor Ontology-based Learning Environments," presented at International Conference Computers in Education (ICCE 2002), Auckland, New Zealand, 2002.

[26] V. Devedzic, L. Jerinic, and D. Radovic, "The GET-BITS Model of Intelligent Tutoring Systems," *Interactive Learning Research (JILR)*, vol. 11, pp. 411-434, 2000.

[27] N. Noy and C. Hafner, "The State of the Art in Ontology Design," in *AI Magazine*, vol. 18, 1997, pp. 53-74.

[28] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, pp. 199 - 220, 1993.

[29] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins, "What Are Ontologies, and Why Do We Need Them?," *IEEE Intelligent Systems*, vol. 14, pp. 20-26, 1999.

[30] J. Kay and S. Holden, "Automatic Extraction of Ontologies from Teaching Document Metadata," presented at International Conference on Computers in Education (ICCE 2002), Auckland, New Zealand, 2002.

[31] S. Cranefield and M. Purvis, "UML as an ontology modelling language," presented at Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99), 1999.

[32] S. Cranefield, "UML and the Semantic Web," International Semantic Web Working Symposium (SWWS), 2001.

[33] L. Heaton, "XML Metadata Interchange (XMI), v2.0," Object Management Group (OGM) formal/03-05-02, 5 March 2003.

[34] D. Hartley and A. Mitrovic, "Supporting learning by opening the student model," presented at 6th Int. Conf on Intelligent Tutoring Systems, Biarritz, France, 2002.

[35] B. Martin, "Constraint-Based Modelling: Representing Student Knowledge," *New Zealand Journal of Computing*, vol. 7, pp. 30-38, 1999.

[36] A. Weerasinghe and A. Mitrovic, "Effects of self-explanation in an open-ended domain," presented at Artificial Intelligence in Education (AIED 2003), 2003.

[37] G. D. Penna, A. D. Marco, B. Intrigila, I. Melatti, and A. Pierantonio, "Xere : Towards a Natural Interoperability between XML and ER Diagrams," presented at Fundamental Approaches to Software Engineering (FASE 2003), Warsaw, Poland, 2003.

[38] T. Bray, D. Hollander, and A. Layman, "Namespaces in XML," World Wide Web Consortium, Technical Report REC-xml-names-19990114, 14-January 1999.

[39] E. El-Sheikh and J. S. 636-642, "Using Hierarchical Classification-Based Expert Systems to Support Tutoring," presented at International Conference on Artificial Intelligence, Las Vegas, Nevada, USA, 2003.

[40] B. Martin and A. Mitrovic, "Automatic Problem Generation in Constraint-Based Tutors," presented at Sixth International Conference on Intelligent Tutoring Systems, Biarritz, 2002.

[41] A. Munro, M. C. Johnson, Q. A. Pizzini, D. S. Surmon, D. M. Towne, and J. L. Wogulis, "Authoring simulation-centered tutors with RIDES," *Artificial Intelligence in Education*, vol. 8, pp. 284-316, 1997.

[42] B. Martin and A. Mitrovic, "Learning Domain Constraints by Asking Questions," presented at ITS'2000 workshop on applying Machine Learning to ITS Design/Construction, Montreal, 2000.

[43] C. Sammut and R. Banerji, "MARVIN: Learning concepts by asking questions," presented at Machine Learning: An Artificial Intelligence Approach, Los Altos, California, 1986.

[44] B. P. Woolf, J. Reid, N. Stillings, M. Bruno, D. Murray, P. Reese, A. Peterfreund, and K. Rath, "A General Platform for Inquiry Learning," presented at 6th International Conference in Intelligent Tutoring Systems (ITS 2002), Biarritz, France and San Sebastian, Spain, 2002.

[45] D. Abraham and K. Yacef, "XMLTutor - an Authoring Tool for Factual Domains," presented at International Conference on Computers in Education, Auckland, New Zealand, 2002.

[46] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, and H. F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework," Microsoft - Sun Microsystems - IBM - Canon, W3C Recommendation REC-soap12-part1-20030624, 24 June 2003.

[47] A. Mitrovic, "NORMIT: A Web-Enabled Tutor for Database Normalization," presented at International Conference on Computers in Education (ICCE 2002), Auckland, New Zealand, 2002.

[48] P. Hayes and B. McBride, "RDF Semantics," IHMC - Hewlett Packard Labs, W3C Proposed Recommendation PR-rdf-mt-20031215, 15 December 2003.

[49] D. L. McGuinness and F. v. Harmelen, "OWL Web Ontology Language Overview," Stanford University - Vrije Universiteit, W3C Proposed Recommendation PR-owl-features-20031215, 15 December 2003.

[50] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "DAML+OIL Reference Description," W3C Note 18 December 2001.

[51] J. Hendler, T. Berners-Lee, and E. Miller, "Integrating Applications on the Semantic Web," *Institute of Electrical Engineers of Japan*, vol. 122, pp. 676-680, 2002.

9 Appendix

9.1 ER Domain Schema

This XML schema defines a simple ER domain model. It is a modified version of the ER XML Schema that defines ER diagrams generated using the Xere algorithm[6].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">

  <xs:element name="ER">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="entity"/>
        <xs:element ref="relation"/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name="entity">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="attribute" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:key name="entity_key">
      <xs:selector xpath="."/>
      <xs:field xpath="name"/>
    </xs:key>
  </xs:element>

  <xs:element name="relation">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="attribute" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="optional"/>
      <xs:attribute name="from" type="xs:string" use="required"/>
      <xs:attribute name="to" type="xs:string" use="required"/>
      <xs:attribute name="card" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
              <xs:pattern value="\d+,(\d+ | unbounded) -\d+,(\d+ |
unbounded)"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
    <xs:keyref name="from_ref" refer="entity_key">
      <xs:selector xpath="."/>
      <xs:field xpath="from"/>
    </xs:keyref>
    <xs:keyref name="to_ref" refer="entity_key">
```

[6] The XML schema for ER diagrams was sourced from: http://dellapenna.univaq.it:800/xere/index.asp

```
      <xs:selector xpath="."/>
      <xs:field xpath="to"/>
    </xs:keyref>
   </xs:element>

  <xs:element name="attribute">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="type" type="xs:string" use="required"/>
    </xs:complexType>0
  </xs:element>
</xs:schema>
```

Listing 8 : The full ER XML Schema domain definition for the ER domain.