



UNIVERSITY OF MACEDONIA
DOCTORAL STUDIES PROGRAMME
DEPARTMENT OF APPLIED INFORMATICS

THE MATHESIS META-AUTHORING FRAMEWORK FOR INTELLIGENT
TUTORING SYSTEMS IN MATHEMATICS

Doctoral Thesis

of

Dimitrios Sklavakis

Thessaloniki, 07/2015

THE MATHESIS META-AUTHORING FRAMEWORK FOR INTELLIGENT
TUTORING SYSTEMS IN MATHEMATICS

Dimitrios Sklavakis

B.Sc. in Mathematics, Aristotle University of Thessaloniki, 1993
M.Sc. in Artificial Intelligence, Edinburgh University, 1998

Doctoral Thesis

Submitted in partial fulfilment of the requirements for the

DEGREE OF DOCTOR OF PHILOSOPHY
IN APPLIED INFORMATICS

Supervisor: Dr. Ioannis Refanidis, Associate Professor

Approved by the 7-membered examining committee the 07/07/2015

Dr. Ioannis Refanidis
Associate Professor

Dr. Stefanidis Georgios,
Professor

Dr. Chatzigeorgiou Alexandros
Associate Professor

Dr. Satratzemi Maria
Professor

Dr. Evangelidis Georgios,
Professor

Dr. Samaras Nikolaos
Associate Professor

Dr. Sakellariou Ilias
Lecturer

Dedication

Dedicated to the Teachers, the Students and the Researchers of Knowledge.

Αφιερωμένο στους Δασκάλους, τους Μαθητές και τους Ερευνητές της Γνώσης.

Acknowledgements

I would like to thank my supervisor, Dr. Ioannis Refanidis, for the confidence and patience he showed.

I would like to thank my wife, Kyriaki, for her long-lasting support and patience.

I would like to thank my friend and colleague, Mr. Sotiris Sakellaris, B.Sc, M.Sc. Mathematics, for our enlightening discussions as well as for his efforts in using the MATHESIS framework.

Finally, I would like to thank all the reviewers and editors of my published work for their helpful comments.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα της διατριβής αυτής, Δρ. Ιωάννη Ρεφανίδη, για την εμπιστοσύνη και την υπομονή που επέδειξε.

Θα ήθελα να ευχαριστήσω τη σύζυγό μου, Κυριακή, για την μακρόχρονη υποστήριξη και υπομονή.

Θα ήθελα να ευχαριστήσω τον φίλο και συνάδελφο, κ. Σωτήρη Σακελλάρη, τόσο για τις διαφωτιστικές μας συζητήσεις, όσο και για την προσπάθεια που κατέβαλλε στη χρήση του συστήματος ΜΑΘΗΣΙΣ.

Τέλος, θα ήθελα να ευχαριστήσω όλους τους κριτές και επιμελητές των δημοσιεύσεών μου για τις εποικοδομητικές παρατηρήσεις τους.

Abstract: The effect of the knowledge acquisition bottleneck is still limiting the widespread use of knowledge-based systems (KBS), especially in the area of model-tracing tutors, as they demand the development of deep domain expertise, tutoring and student models. The MATHESIS meta-authoring framework for model-tracing tutors, presented in this thesis, aims at maximizing knowledge reuse. This is achieved through ontological representation of both the declarative and procedural knowledge of a model-tracing tutor (MTT), as well as of the declarative and procedural authoring knowledge of the process to develop a MTT. Declarative knowledge is represented in Ontology Web Language (OWL). Procedural knowledge is represented using the concepts of atomic and composite processes of OWL-S web services description ontology. The framework provides authoring tools, integrated into the Protégé OWL ontology editor, for the development and management of the MTT's ontological representation. It also provides meta-authoring tools for the ontological representation of the authoring expertise as a set of composite authoring processes and atomic authoring statements. The latter constitute a language, ONTOMATH, for building executable authoring models that, when executed by the tools, guide non-expert authors like domain experts to the creation of new model-tracing tutors. The framework, being in an experimental stage, was used for the development of a monomial multiplication and division tutor. However, the overall design and implementation aimed at constituting the framework as a proof-of-concept system that can be used for the meta-knowledge engineering of more complex model-tracing tutors.

Table of Contents

List of Tables	x
List of Figures	xi
Chapter 1: Introduction	17
1.1 The Problem.....	17
1.2 Motivation.....	20
1.3 Contribution.....	22
1.4 Summary of Results.....	24
Chapter 2: The MATHESIS Algebra School.....	28
2.1 Introduction.....	28
2.2 The MATHESIS Algebra Tutor.....	31
2.2.1 The Tutor’s Domain Expertise Model	34
2.2.2 Intelligent Task Recognition.....	39
2.2.3 The Tutoring Model: Deep Model Tracing With Intelligent Task Recognition	44
2.3 The Learning Management System	57
2.4 Related Work	63
2.5 Evaluation of the MATHESIS System	66
2.5.1 Evaluation by Teachers.....	67
2.6 Discussion and Further Work	75
Chapter 3: The MATHESIS Meta-Authoring Framework	79
3.1 Introduction.....	79
3.2 Background	81
3.2.1 Related Work	81
3.2.2 Ontological Engineering and the Knowledge Gap Problem.....	86
3.3 An Overview of the MATHESIS Meta-Authoring Framework	88
3.4 The ONTOMATH Meta-Knowledge Engineering Language.....	93
3.4.1 Procedural Knowledge Representation: The OWL-S Process Model	94

3.4.2 Procedural Authoring Knowledge Representation: The ONTOMATH language	97
3.5 The MATHESIS Authoring Tools	101
3.5.1 The Tutor Authoring Tools	101
3.5.2 The Authoring Processes (Meta-Authoring) Tools.....	107
Chapter 4: Tutor Authoring in the MATHESIS Framework	113
4.1 Introduction.....	113
4.2 Tutor Initialization	114
4.3 Cognitive model initialization.....	117
4.4 Tutoring model initialization	121
4.5 Program code model	124
4.6 Interface model initialization	126
4.7 Execution of Authoring Processes	128
Chapter 5: Discussion	152
5.1 Knowledge reuse and scalability	152
5.2 Conclusions and further work	156
Appendix A: Complete Math Domain Cognitive Model of the MATHESIS Algebra Tutor.....	160
A1. Monomial multiplication:	160
A2. Monomial Division	160
A3. Collection of Like Terms	161
A4. Monomial Power	161
A5. Monomial by Polynomial Multiplication.....	162
A6. Polynomial by Polynomial Multiplication.....	162
A7 Parentheses' Elimination.....	162
A8. Square of Sum/Difference Expansion.....	163
A9. Product of Sum by Difference Expansion.....	164
A10. Cube of Sum/Difference Expansion	164
A11. Factoring by Common Factor	166
A12 Factoring by Difference of Squares	167

A13. Factoring by Sum of Cubes.....	167
A14. Factoring by Difference of Cubes.....	169
A15. Factoring by Square of Sum/Difference	170
A16. Factoring the Quadratic Form.....	170
A17. Factoring by Term Grouping	171
Appendix B: The ONTOMATH Atomic Authoring Statements Reference	174
B1. OntoMath_Browse Statements.....	174
B2. OntoMath_Collection Statements	174
B3. OntoMath_String Statements	175
B4. OntoMath_Dialog Statements	175
B5. OntoMath_Ontology_Editing Statements	176
B6. OntoMath_Tutoring_Processes_Editing Statements.....	179
B7. OntoMath_Ontology_Predicates	179
References.....	180

List of Tables

Table 2.1. Expanding $-10(x-1)(x + 1)$ in three different ways	38
Table 2.2. Alternative Path for Factoring $4x*(x + 7) + 48$	55
Table 2.3. The Fine-Grained Student Model: Solution Steps	56
Table 2.4. Performance of skill “Calculate common factor”. The percentage is $2/4=50%$	57
Table 2.5. Evaluation results given by forty (40) math teachers after a three-hour hands-on workshop (questions are translated from Greek)	69
Table 2.6. Evaluation results given by twenty (20) students after a three-month period (questions are translated from Greek)	71
Table 2.7. Students’ performance rise by the MATHESIS Algebra Tutor	73
Table 3.1. Common control constructs supported by the OWL-S process model.	96
Table 3.2. The ONTOMATH Statements and their operations	99

List of Figures

Fig. 2.1. The MATHESIS Algebra Tutor Interface.....	32
Fig. 2.2. MathML Presentation code for expression $4x*(x + 7) + 48$ before and after intelligent task recognition.....	40
Fig. 2.3 Mathematical objects created by intelligent task recognition for expression $4x*(x + 7) + 48$	41
Fig. 2.4. The student proposes the operation “FACTORING-Common Factor” from the drop-down list of supported operations to be applied to the selected expression.	46
Fig. 2.5. The tutor checks and confirms the student’s suggested operation “Common Factor” through messages 2.1 and 2.2 (top). The common factor under question here is 4, denoted by the empty square scaffold in the “ANSWERING SPACE” area (bottom right).	47
Fig. 2.6. The tutor confirms the entered common factor and asks for the first quotient by messages 2.3 and 2.4 (top). The quotient under question is $4x*(x + 7): 4 = x*(x + 7)$ denoted by the $\square^\square * (\square)^\square$ scaffold in the “ANSWERING SPACE” area (right).....	48
Fig. 2.7. The tutor confirms the first quotient and asks for the second quotient through messages 2.7 and 2.8 (top). The quotient under question is $48: 4 = 12$ denoted by the empty square scaffold in the “ANSWERING SPACE” area (right).....	49
Fig. 2.8. Successful completion of the common factor method in expression $4x*(x + 7) + 48$	50

Fig. 2.9. Successful completion of the monomial-polynomial multiplication $x \cdot (x + 7)$	51
Fig. 2.10. First step of factoring $x^2 + 7x + 12$. The student must identify $a \cdot b = P = +12$ and $a + b = S = +7$	52
Fig. 2.11. Responding to a student error. The tutor displays an error message, gives help (top, message 6.4) and asks for the correct answer (right).	53
Fig. 2.12. Successful completion of factoring $4x \cdot (x + 7) + 48$	53
Fig. 2.13. The Student Model: Skill Performance Statistics.....	57
Fig. 2.14. The Teachers' Menu.....	58
Fig. 2.15. The Classes Management Page.	58
Fig. 2.16. Test Paper Editing. The author has just created exercise no. 22 using the HTML editor (b) and inserted expression $4x \cdot (x + 7) + 48$ for the first question using the math editor (c). The paper is shown on the right with the newly added exercise at the bottom (d).	60
Fig. 2.17. Individualized Assignment of Exercises to Students.	61
Fig. 2.18. Student Assessment: Selecting a Solved Exercise	62
Fig. 3.1 The MATHESIS Meta-Authoring Framework.....	92
Fig. 3.2. The MATHESIS Tools as a tab widget in Protégé: (a) Framework-specific (model-tracing) Tutor Authoring Tools, (b) Authoring Processes (Meta-Authoring) Tools, (c) The MATHESIS Ontology Tab.....	93
Fig. 3.3. Top level of the OWL-S <i>process</i> ontology (from Martin et al., 2005)....	96
Fig. 3.4. Part of the ONTOMATH Authoring Processes Ontology.....	98
Fig. 3.5 The Tutor Initialization Tools.....	101
Fig. 3.6 The XML DOM tree of the MATHESIS Algebra Tutor interface.....	103
Fig. 3.7 The Tutoring Processes Advanced Authoring Tools.....	104

Fig. 3.8 A newly created Tutoring Process.....	104
Fig. 3.9 The Calling Sequence Tree for Tutoring Process multiplyMainParts ..	105
Fig. 3.10 The Authoring Processes (Meta-Authoring) Tools	108
Fig. 4.1. The Model-Tracing Tutor Authoring Tools Window: (a) The Tutor Initialization Tools, (b) The Advanced Tools for Tutoring Processes Authoring, (c) Tree representation of tutoring process Model-Tracing- Algorithm for the execute-monomial-multiplication task.....	114
Fig. 4.2. The top-level ontological representation of the tutor	115
Fig. 4.3. Author is prompted by the tools to enter the name of a newly created tutor instance.	116
Fig. 4.4. (a) The ITS_Implemented hierarchy (b) Instance monomial- multiplication-tutor is selected (c) Properties of the selected instance	116
Fig. 4.5. Author is prompted by the tools to enter the name of a newly created cognitive task instance.	117
Fig. 4.6. (a) The Domain_Task hierarchy (b) Instance execute-monomial- multiplication is selected (c) Properties of the selected instance .	118
Fig. 4.7. (a) The ITS-Teaching-Model hierarchy. (b) Instance execute-monomial- multiplication-Model-Tracing-Algorithm has been selected. (c) Properties of the selected instance are shown.....	119
Fig. 4.8. (a) The Domain-Knowledge-Component hierarchy. (b) Instance monomial has been selected. (c) Properties of the selected instance are shown.	120
Fig. 4.9. Representation of the JavaScript function multiplyMainParts	124
Fig.4.10 Part of the JavaScript_Statement ontology	125

Fig. 4.11. The HTML User Interface DOM Ontological (left) and Visual (right, top) Representation.....	127
Fig. 4.12. The Authoring Processes Authoring (Meta-Authoring) Tools displaying Authoring Process <code>authoring_task_present_domain_task</code>	129
Fig. 4.13. The <code>identify_input_knowledge_components</code> authoring process....	132
Fig. 4.14. Locating a monomial instance in the ontology.....	134
Fig. 4.15. Creating a new instance of monomial	135
Fig. 4.16 Instance <code>currentAuthoringSession</code> for the monomial-multiplication-tutor	136
Fig. 4.17. The <code>define-interface-elements-for-input-knowledge-components</code> authoring process	137
Fig. 4.18 The <code>add-interface-element-to-DOM</code> authoring process	139
Fig. 4.19 Ontological representation of a monomial tutor with its user interface	141
Fig. 4.20 Authoring process <code>define-variables-for-interface-elements</code>	142
Fig. 4.21 The <code>expressionInputControl</code> JavaScript variable	143
Fig. 4.22 The <code>define_code_to_initialize_interface_elements</code> authoring process	144
Fig. 4.23 The <code>getHTMLElementProperty</code> authoring process	145
Fig. 4.24 Ontological representation of JavaScript statement <code>expressionInputControl=getElementById(“expressionInputControl”)</code>	147
Fig. 4.25 Authoring process <code>get_interface_element_reference</code>	148
Fig. 4.26 Tutoring process <code>execute-monomial-multiplication-Presentation</code> .	149

CHAPTER 1



Chapter 1: Introduction

1.1 THE PROBLEM

The main goal of this thesis is the development of an ontology-based authoring framework for the development of model-tracing tutors (MTT) for mathematics. The purpose of the framework is to encode the knowledge of expert authors of MTTs and make it available and reusable to other authors, either equally or less expert. This framework is called MATHESIS, the Greek word for “learning”, the root of the word “mathematics”.

Intelligent tutoring systems (ITS), particularly model-tracing tutors, have been proven quite successful in the area of mathematics (Koedinger, Anderson, Hadley, & Mark, 1997; Koedinger & Corbett, 2006). Despite their efficiency (Corbett 2001), these tutors are expensive to build both in time and human resources (Alevan, McLaren, Sewall, & Koedinger, 2006). This is due to the well-known *knowledge acquisition bottleneck* (Hoffman 1987), comprising the extraction of knowledge from domain

experts, the representation of this knowledge and its implementation in effective knowledge-based systems.

Knowledge acquisition and its counterpart, knowledge reuse, have been proven to be the key problems for the development of expertise models, the models that represent and produce the problem-solving knowledge in knowledge-based systems. The main consequences are:

- High development demands in human resources, time and money.
- Demand for knowledge engineers possessing significant expertise.
- Shallow, incomplete or even incorrect expertise models.
- Difficulties in modifying and/or expanding the expertise models.
- Inability to reuse developed expertise models in similar or new knowledge-based systems (an effect described as “re-inventing the wheel”).

In the case of MTTs, the knowledge acquisition bottleneck gets even more serious as these systems must contain two expertise models:

- i. The *domain expertise model* or *problem solver*, which represents the problem-solving knowledge of the tutored domain. This model is used to produce the valid solution steps of the tutored problem and allow the tutor to provide guidance and feedback to the student.
- ii. The *pedagogical* or *tutoring model*, which represents the teaching knowledge of the system such as how to present the problem, what problem-solving tools to provide to the students for entering their solution steps, when and how to give help, what kind of help/guidance to give etc.

In turn, these models affect directly the design of the *user interface model*, which orchestrates the interactions between the aforementioned two models to produce the desired tutoring behaviour. In addition, some MTTs require the development of another model, the *student model*, which represents students' mastery of the tutored domain. This model is used by the system to provide student-adapted tutoring either within problems (micro-adaptation) or between problems (macro-adaptation).

The most difficult model to build is the domain expertise model. At the same time, it is the most critical one since it defines:

- i. The tutor's *breadth*, that is, how many domain skills it can teach.
- ii. The tutor's *depth*, that is, how complex skills, in terms of the sub-skills contained, it can teach.
- iii. The tutor's *granularity*, that is, how fine-grained are the solution steps that the tutor can produce and guide.
- iv. The tutor's *scalability*, that is, the ability to reuse the tutor's domain expertise model for extending its breadth and depth.

Despite the efforts, advancements and successes in the currently developed authoring frameworks and the corresponding tutors, these frameworks have worked around the knowledge acquisition problem rather than confronting it directly. As a consequence, most of the developed tutors suffer from limited depth and breadth, whereas those having broader and deeper domain expertise models suffer from scalability issues. The motivation of this thesis is to develop an authoring framework that will deal directly with the knowledge acquisition problem in order to produce tutors that cover broader and more complex domains in a scalable way.

1.2 MOTIVATION

In an extensive survey of authoring tools, Murray (2003a) concluded that they suffered from a number of problems such as isolation, fragmentation and lack of communication, interoperability and re-usability of the tutors they build. The same problems had been identified three years earlier in (Mizoguchi & Bourdeau, 2000). These problems are not specific to the domain of ITS authoring, as they penetrate the whole area of expert systems development (Lenat & Guha, 1990; Lenat, 1995). A highly promising solution to all of them is *ontological engineering*, that is, the development of ontologies that represent declaratively the expertise that lies inside any intelligent system (Mizoguchi, 2004). The main advantages of ontologies are that:

- i. They impose a systematic and structured development of knowledge, just like developing a mathematical theory with definitions, properties, axioms and theories, and
- ii. The developed knowledge, being in a declarative form, is open for inspection and therefore mostly reusable (Gómez-Pérez, Fernández-López, Corcho, 2004).

Based on the success of the ontological engineering approach in the domain of expert systems (Aitken & Sklavakis, 1999; Lenat, 1995; Sklavakis, 1998), as well as in the domain of intelligent tutoring systems (Mizoguchi, Hayashi, & Bourdeau, 2009), two research goals were set:

- i. The complete ontological representation of a model-tracing tutor's modules, that is, the user interface, the tutoring model, the domain expertise model, the student model, as well as of the authoring knowledge that was used to build these models, and
- ii. The extensive use of standardized languages and publicly available modular tools.

For these reasons, a bottom-up approach was adopted: Initially, the MATHESIS Algebra Tutor was developed to be used as a prototype target tutor (Sklavakis & Refanidis, 2008; Sklavakis & Refanidis 2013). It is a model-tracing tutor that teaches expansion and factoring of algebraic expressions. Having knowledge reuse as its primary design guidelines, the tutor is implemented using HTML for the user interface and JavaScript for the domain expertise and tutoring models. The primary interface element is Design Science's WebEq Input Control applet¹, an editor for displaying and editing mathematical expressions. The WebEq Input control is scriptable through JavaScript and uses MathML² to represent algebraic expressions. The tutor has a cognitive model of considerable breadth, depth and granularity, easily scalable. Then, based on the knowledge used to develop the MATHESIS Algebra Tutor, an initial version of the MATHESIS ontology has been developed using the Ontology Web Language - OWL³ (Sklavakis, & Refanidis, 2010b). The ontology was developed using the Protégé ontology editor⁴. As this first version of the ontology was developed in a bottom-up direction, it emphasized on the representation of the tutor's models, namely the interface, tutoring and domain expertise models. The ontology also contained a representation of the authoring knowledge at a rather conceptual level. At the final stage, generic meta-authoring tools were developed (Sklavakis & Refanidis, 2014). These tools include:

- i. An executable authoring language, ONTOMATH, based on the process model of OWL-S⁵,

¹ <http://www.dessci.com/>

² <http://www.w3.org/Math/>

³ <http://www.w3.org/TR/owl-features/>

⁴ <http://protege.stanford.edu>

⁵ <http://www.w3.org/Submission/OWL-S/>

- ii. Editing tools for the development of ONTOMATH executable authoring expertise models, that is, an ontological representation of the declarative and procedural authoring knowledge, and
- iii. An interpreter for executing the ONTOMATH authoring models.

These tools constitute the MATHESIS authoring framework as a *meta-authoring framework*. In an authoring framework the tools allow expert authors to directly develop the various models of a tutor. In the MATHESIS meta-authoring framework expert authors, using the ONTOMATH language, build executable authoring models that encode their authoring knowledge of how to build a tutor. When these authoring models are executed by non-expert domain authors they guide them in developing the ontological representations of the tutors' models. These ontological representations are automatically translated into program code that implements the tutors.

Using these tools, an authoring model was developed that, when executed by the interpreter, guides a trained domain author (teacher of mathematics) to build the ontological representation of a model-tracing monomial multiplication tutor identical to the one contained in the original MATHESIS Algebra Tutor. In parallel, special authoring tools have been developed for the authoring of model-tracing tutors. These tools are used to support the meta-authoring tools in the development of the executable authoring model by automating some top-level authoring processes of the MTT under development and providing visualization and browsing facilities for the inspection of the tutor's developed models. All authoring tools were developed as a tab widget in Protégé using Java.

1.3 CONTRIBUTION

During the research for this thesis, the following contributions have been made:

A. Publications and System Demonstrations

Sklavakis, D., & Refanidis, I. (2008). An Individualized Web-Based Algebra Tutor Based on Dynamic Deep Model-Tracing. *Proceedings of the Fifth Hellenic Conference on Artificial Intelligence (SETN '08)*, (pp. 389-394). Heidelberg: Springer.

Sklavakis, D. & Refanidis, I. (2009a). The MATHESIS Algebra Tutor: Web-based Expert Tutoring via Deep Model Tracing. Interactive Event. *Proceedings of the 14th International Conference on Artificial Intelligence in Education (AIED 2009)*, (p. 795). Amsterdam: IOS Press.

Sklavakis, D., & Refanidis, I. (2009b). The MATHESIS Ontology: Reusable Authoring Knowledge for Reusable Intelligent Tutors. *Proceedings of the 7th International Workshop on Ontologies and Semantic Web for E-Learning (SWEL 2009)*, (pp. 86-90).

Sklavakis, D., & Refanidis, I. (2010a). MATHESIS: A Web-Based Intelligent Tutoring School for Algebra. *Intelligent System Demonstration at the 6th Hellenic Conference on Artificial Intelligence (SETN 2010)*.

Sklavakis, D., & Refanidis, I. (2010b). Ontology-Based Authoring of Intelligent Model-Tracing Math Tutors. *Proceedings of the Fourteenth International Conference on Artificial Intelligence (AIMSA 2010)*, (pp. 201-210). Heidelberg: Springer.

Sklavakis, D., & Refanidis, I. (2013). MATHESIS: An Intelligent Web-Based Algebra Tutoring School. *International Journal of Artificial Intelligence in Education* Vol. 22 (2) (pp. 191-218). Amsterdam: IOS Press.

Sklavakis, D., & Refanidis, I. (2014). The MATHESIS meta-knowledge engineering framework: Ontology-driven development of intelligent tutoring systems. *Applied Ontology* Vol. 9 (3-4) (pp. 237-265). Amsterdam: IOS Press.

B. Software

- The MATHESIS intelligent Algebra Tutoring System (Section 2) (http://users.sch.gr/dsklavakis/mathesis/en/MATHESIS_Main_Frameset.htm)
- The MATHESIS Authoring Tools (Sections 3 and 4) (http://ai.uom.gr/dsklavakis/en/mathesis/kes2011/01-Authoring_Tools.mp4) and (http://ai.uom.gr/dsklavakis/en/mathesis/kes2011/02-Authoring_Processes.mp4)
- The MATHESIS ontology (Sections 3 and 4)

1.4 SUMMARY OF RESULTS

Two are the main results of this thesis, one concerning the MATHESIS Algebra Tutor and the other the MATHESIS meta-authoring framework.

First, the MATHESIS Algebra Tutor is a successful proof-of-concept system (Skavakis & Refanidis, 2013). The basic research result is that, in order to build successful intelligent real-world tutoring systems, we must build powerful domain expertise models. The engineering of such broad and deep models has to overcome the common obstacle of all expert systems, the knowledge acquisition bottleneck: the extraction of the expertise from domain experts and its representation in efficient ways. In the domain of knowledge engineering, the most profitable solution up to now is knowledge reuse, which is achieved through open, modular, interchangeable, inspectable, formal knowledge representations and system implementations (Aitken & Sklavakis 1999). Equally important, the models must be deep and broad, having a wide basis of low level knowledge about simple task performance, on top of which is built the knowledge for performing higher level domain tasks. Otherwise, models are brittle (Lenat & Guha 1990), performance is limited, scaling up is intractable and the systems fail to cope with real-world demands. The MATHESIS Algebra Tutor incorporates all these characteristics that make it a successful real-world intelligent tutoring system.

Second, the MATHESIS meta-authoring framework achieves the development of broad, deep, granular and scalable authoring models. It allows the ontological representation of expert authoring knowledge in an arbitrary breadth, depth and granularity in the form of executable authoring processes. Thus, it makes MTTs' authoring scalable by spreading its load over various levels of reusable authoring processes and over various authors, experts and non-experts, that can reuse them by browsing, locating and modifying them (Sklavakis & Refanidis, 2014).

In Chapter 2 the MATHESIS Algebra School is described with emphasis on the MATHESIS Algebra Tutor around which the school is built. Chapter 3 describes the MATHESIS meta-authoring framework and its constituent parts. Related work is presented separately for the MATHESIS Algebra Tutor (Section 2.4) and separately for the MATHESIS framework (Section 3.2). Chapter 4 describes how the framework was used to develop a monomial multiplication model-tracing tutor. Finally, Chapter 5 discusses the results of the research as well as further research directions.

Chapter 2

Chapter 2: The MATHEISIS Algebra School

2.1 INTRODUCTION

One-to-one tutoring has proven to be one of the most effective ways of teaching. It has been shown (Bloom 1984) that the performance of the average student under an expert tutor is about two standard deviations above the average performance of the conventional class (30 students to one teacher). That is, 50% of the tutored students scored higher than 98% of students in the conventional class. However, it is also known that one-to-one tutoring is a very expensive form of education. Due to this cost, we are still in the era of mass education, struggling to raise the teacher to student ratio. The problem of designing and implementing educational environments as effective as individual tutoring was termed by Bloom as “the two sigma problem”, named after the mathematical symbol of standard deviation, σ .

The implementation of the one-to-one tutoring model by Intelligent Tutoring Systems (ITSs) has motivated researchers to aim to develop ITSs that provide the same tutoring quality as a human tutor (VanLehn 2006). Model Tracing Tutors (MTTs)

(Anderson, Corbett, Koedinger, & Pelletier, 1995) have shown significant success in domains like mathematics (Koedinger & Corbett 2006), computer programming (Corbett 2001) and physics (VanLehn, Lynch, Schulze, Shapiro, & Shelby, 2005). These tutors are based on a domain expertise model that solves the problem under tutoring and produces the correct step(s). At each step, the model-tracing algorithm matches the solution(s) produced by the model to that provided by the student and gives positive or negative feedback, hints or/and help messages. However, the domain models of MTTs are hard to author (Alevan, McLaren, Sewall, & Koedinger, 2006). The main reason for this is the knowledge acquisition bottleneck: extracting the knowledge from the domain experts and encoding it into a MTT. Knowledge reuse has been proposed as a key factor to overcome this obstacle (Murray 2003a; Mizoguchi & Bourdeau 2000). Since expert knowledge and, particularly, tutoring knowledge is so hard to create, re-using it is of paramount importance. A good example of knowledge reuse is the Mass Production mechanism provided by Carnegie Mellon's Cognitive Tutors Authoring Tools (CTAT). This mechanism allows the creation of new tutors from existing ones for isomorphic problems, that is problems having nearly the same solution steps (Alevan, McLaren, & Sewall, 2009).

The main goal of this thesis is to develop authoring tools for model-tracing tutors in mathematics, with knowledge re-use as the primary characteristic of the authored tutors as well as for the authoring knowledge used by the tools. For this reason, in the first stage of the MATHESIS project, an Algebra Tutor was developed to be used as a prototype target tutor (Sklavakis & Refanidis 2008; Sklavakis & Refanidis 2013). The purpose of developing the tutor was twofold: a) to investigate the design and implementation effort for developing an MTT having a domain expertise model with a breadth of 16 top level skills (algebraic operations) and – after elaborate cognitive task

analysis – a greater depth and b) to provide the knowledge that would be represented in an ontology on top of which the authoring tools would be implemented (Sklavakis & Refanidis 2009b; Sklavakis & Refanidis 2010b; Sklavakis & Refanidis 2014).

Concerning the former research goal, as the domain expertise model has been extended and deepened, the *scaling-up* problem was confronted: if a problem contains more than one task to be performed then a more complex task arises, i.e., identifying the tasks to perform! The solution to this tutoring problem was to equip the tutor with *intelligent task recognition* through sophisticated parsing of the algebraic expressions. Another, rather positive, consequence of adopting a broad and deep domain expertise model was the development of an equally detailed student model. Instead of simply keeping a percentage measure of the students' skill performance, the student model was extended to keep full records of the interactions between the interface and the student for each solution step.

This chapter describes the web-based intelligent MATHESIS Algebra Model Tracing Tutor as well as the MATHESIS tutoring school for expanding and factoring algebraic expressions. The school has been built around the MATHESIS algebra MTT and has been extended with a learning management system (LMS). The rest of the chapter is structured as follows: Section 2.2 describes the final version of the MATHESIS algebra MTT with an extended domain model, a refined student model and a new interface integrating the tutor into the school. Section 2.3 describes the learning management system of the school, including an editor for teachers to create test papers with their own exercises and tools to inspect the student model. Section 2.4 presents related work. Section 2.5 presents an evaluation of the system while Section 2.6 concludes the chapter with a discussion of the research results and future directions of research.

2.2 THE MATHESIS ALGEBRA TUTOR⁶

The MATHESIS Model-Tracing Algebra Tutor was developed as a prototype target tutor for the MATHESIS project (Sklavakis & Refanidis 2008; Sklavakis & Refanidis 2013). The ultimate goal of the project is the development of authoring tools for model-tracing tutors that will make extensive reuse of the valuable tutoring knowledge through ontological engineering. The MATHESIS tutor itself was designed with knowledge reuse as its main non-functional requirements. Consequently, the architecture of the system should be based on open, standardized and modular representations. Additionally, there were three more issues that determined the overall architecture:

- i. The tutor interface should be web-based in order to be broadly accessible.
- ii. The model-tracing algorithm requires constant interaction between the cognitive model and the interface. Therefore they should lie at the same side, that is, the client side.
- iii. The programming language(s) that would implement the various tutor parts (interface, domain model) should be simple enough to be represented with an ontology. This ontology would be used by the authoring tools to guide non-expert authors in redeveloping the tutor.

The achievement of these requirements led to an implementation of the tutor using HTML for the user interface and JavaScript for the domain expertise and tutoring models. These two languages are the simplest ones for building dynamic, interactive web pages, they are open, non-proprietary and lend themselves to direct representation and manipulation from the developed MATHESIS authoring tools (Sklavakis & Refanidis

⁶ http://users.sch.gr/dsklavakis/mathesis/en/MATHESIS_Main_Frameset.htm

2010b; Sklavakis & Refanidis 2014). The user interface, shown in Figure 2.1, has four main parts:

- i. The messages area (top), where the tutor displays information about the interface usage, as well as hints, help and feedback for correct and incorrect problem-solving steps.
- ii. The algebraic expression rewriting area (a), where the algebraic expression under rewriting and its transformations are displayed.
- iii. The student's answering area (b), where the student enters the answer for each problem-solving step.
- iv. The performed operation area (c), where intermediate results are shown for multi-step algebraic operations.

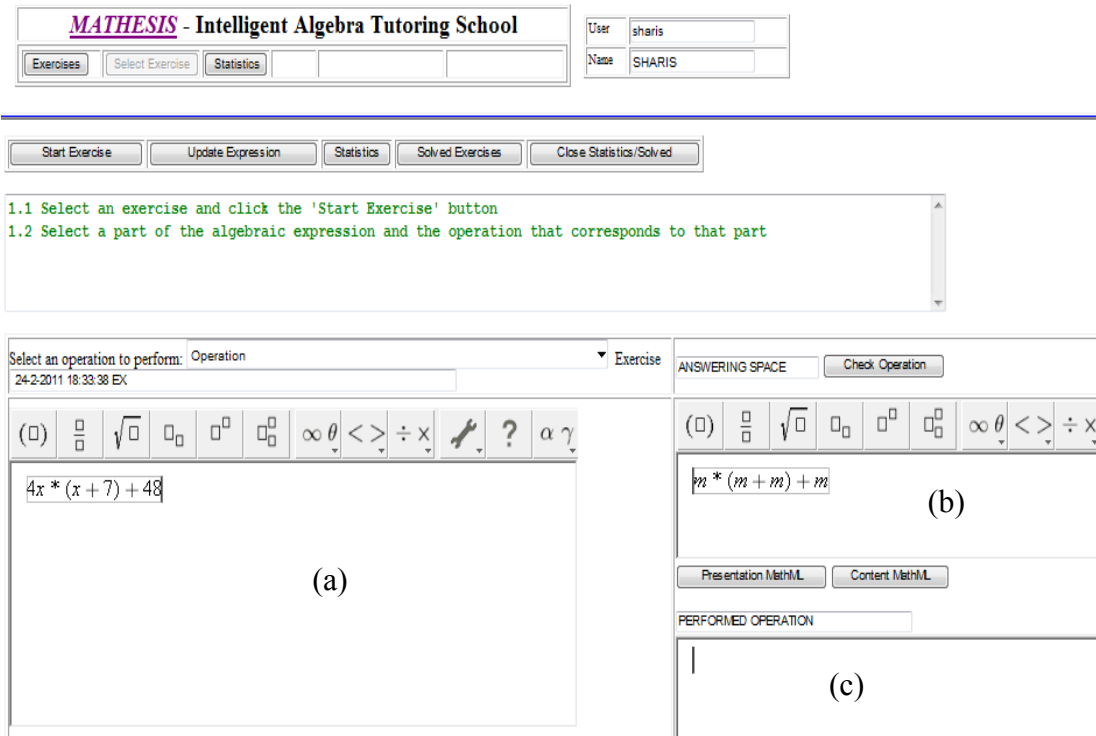


Fig. 2.1. The MATHESIS Algebra Tutor Interface.

The primary interface element is Design Science's WebEq (now MathFlow) Input Control applet, an editor for displaying and editing mathematical expressions in web pages (Design Science 2011). There are three such Input Controls, i.e., the algebraic expression, the answering space and the performed operation Input Controls (Figure 2.1). The WebEq Input Control is scriptable through JavaScript and represents algebraic expressions as MathML . So, during the problem solving process, the problem-solving state as well as the student solution steps are represented via the open MathML standard and, therefore, they can be *interoperatable*, i.e. *inspectable*, *recordable* and *scriptable* (Murray 2003b). As a result, the tutor can be used in the following ways:

- i. The student can type directly in the algebraic expression area algebraic expressions using the math editing palette (Figure 2.1, area (a)). Then, he/she can initialize the tutoring process by clicking the "Start Exercise" button.
- ii. The student can select an exercise from a test paper created by a teacher through the Learning Management System (Section 2.3) and then initialize the tutor.
- iii. The tutor can be initialized (opened) from any other e-learning program with the desired algebraic expression.
- iv. The tutor can recursively initialize (open) new instances of itself in order to break down more complex tutoring tasks.

This latter possibility is directly related to the issues of knowledge re-use and "scaling-up". The mathematical skill of *factoring by term grouping* is rather complex. In this factoring method (a) the terms of the expression must be separated into groups, (b) each group must be factored by some factoring method and (c) the resulting products must have a common factor. It is step (c) that makes step (a) and the whole method

complex and raises the issues of knowledge re-use and “scaling-up”. The intelligent task recognition of the MATHESIS tutor does not yet support guidance for the first step and therefore term grouping is not yet part of its domain model. However, provision has been made for steps (b) and (c). As an example, let’s consider factoring the expression $x^4 - 1 + x^3 - x$ by grouping its terms: the first group, $x^4 - 1$, must be factored using the *identity* $a^2 - b^2 = (a + b)(a - b)$, yielding $(x^2 + 1)(x^2 - 1)$; the second group, $x^3 - x$, must be factored by *common factor*, yielding $x(x^2 - 1)$. To guide the student in applying different factoring methods, the tutor can open an instance of itself with the expression $x^4 - 1$ for the first group followed by an instance for expression $x^3 - x$. Each instance of the tutor can guide the student in factoring each group as separate problems and then return the factored expression to the parent tutor, thus yielding $(x^2 + 1)(x^2 - 1) + x(x^2 - 1)$. From this point, the parent tutor will guide the student in applying the common factor method, yielding $(x^2 - 1)(x^2 + 1 + x)$. Thus, the factoring methods supported by the tutor can be re-used in a completely new and complex factoring task, term grouping.

2.2.1 The Tutor’s Domain Expertise Model

The development of the domain expertise model was based on deep cognitive task analysis in the paradigm of Carnegie-Mellon’s cognitive tutors (Anderson et al. 1995). The tutor can teach a breadth of 16 top-level cognitive math skills:

- Monomial multiplication
- Monomial division
- Powers of monomials
- Monomial-polynomial multiplication

- Polynomial multiplication
- Elimination of parentheses
- Collection of like terms
- Identities expansion: square of sum, square of difference, product of sum by difference, cube of sum and cube of difference
- Factoring: common factor, identities, quadratic form

Each one of these top-level math skills is further analyzed in more detailed sub-skills leading to a fine grained domain model of 104 primitive math skills (see Appendix A). Part of this broad and deep domain model is given in the following list:

1. *Monomial multiplication:* $3x^2y \cdot (-4xz^3) = -12x^3yz^3$
 - 1.1. Multiply coefficients: $3 \cdot (-4) = -12$
 - 1.2. Multiply main parts:
 - 1.2.1. Add exponents of common variables: $x^2 \cdot x = x^{2+1} = x^3$
 - 1.2.2. Copy exponents of single variables: $y \cdot z^3 = yz^3$
2. *Monomial division:* $\frac{-12x^3yz^3}{3x^2y} = -4xz^3$
 - 2.1. Divide coefficients: $-12:3 = -4$
 - 2.2. Divide main parts:
 - 2.2.1. Subtract exponents of common variables: $x^3:x^2 = x^{3-2} = x$ and $y:y = y^{1-1} = y^0 = 1$
 - 2.2.2. Copy exponents of single variables: $z^3 = z^3$
3. *Collection of like terms:* $2xy - x^2 + 7xy + 6x^2 = 9xy + 5x^2$

3.1. Find groups of identical terms: $2xy + 7xy$ and $-x^2 + 6x^2$

3.2. Add the coefficients of each group: $2 + 7 = 9$ and $-1 + 6 = 5$

3.3. Keep the main part of each group: $2xy + 7xy = 9xy$ and $-x^2 + 6x^2 = 5x^2$

4. *Monomial power:* $(-2x^2yz^3)^3 = -8x^6y^3z^9$

4.1. Raise the coefficient to the power: $(-2)^3 = -8$

4.2. Raise main part to the exponent:

4.2.1. Multiply the exponents: $(x^2yz^3)^3 = x^{2 \cdot 3}y^{1 \cdot 3}z^{3 \cdot 3} = x^6y^3z^9$

5. *Monomial by polynomial multiplication:* $3x^3y \cdot (2x - y^2z) = 6x^4y - 3x^3y^3z$

5.1. Identify the monomial terms of the polynomial: $2x$ and $-y^2z$

5.2. Multiply each one of them with the monomial:

$$3x^3y \cdot 2x = 6x^4y \text{ and } 3x^3y \cdot (-y^2z) = -3x^3y^3z$$

6. *Polynomial by polynomial multiplication:*

$$(3xy - 2x^2) \cdot (2x^2y^2 - 4xy) = 6x^3y^3 - 12x^2y^2 - 4x^4y^2 + 8x^3y$$

6.1. Identify the monomial terms of the first polynomial: $3xy$ and $-2x^2$

6.2. Identify the monomial terms of the second polynomial: $2x^2y^2$ and $-4xy$

6.3. Multiply each term of the first monomial with each term of the second

monomial: $3xy \cdot 2x^2y^2 = 6x^3y^3$ and $3xy \cdot (-4xy) = -12x^2y^2$ and $-2x^2 \cdot$

$2x^2y^2 = -4x^4y^2$ and $-2x^2 \cdot (-4xy) = 8x^3y$

7. *Elimination of parentheses:*

$$(5x^2 - 8xy + 3) - (x^2 + 4xy - 5) =$$

$$5x^2 - 8xy + 3 - x^2 - 4xy + 5 =$$

$$4x^2 - 12xy + 8$$

7.1. Keep the sign of each parenthesized term if the sign in front of the parenthesis is

a plus (+): $(5x^2 - 8xy + 3) = 5x^2 - 8xy + 3$

7.2. Change the sign of each parenthesized term if the sign in front of the parenthesis

is a minus (-): $-(x^2 + 4xy - 5) = -x^2 - 4xy + 5$

7.3. Collect like terms if there are any: $5x^2 - 8xy + 3 - x^2 - 4xy + 5 = 4x^2 -$

$12xy + 8$

8. *Identity expansion:* $(2x + 3)^2 = 4x^2 + 12x + 9$

8.1. Recall the expanded form of the identity: $(a + b)^2 = a^2 + 2ab + b^2$

8.2. Substitute a and b for the real terms: $a = 2x$ and $b = 3$

8.3. Take care for parenthesized terms: $(2x + 3)^2 = (2x)^2 + 2 \cdot 2x \cdot 3 + 3^2$

8.4. Perform monomial multiplications and powers: $(2x)^2 + 2 \cdot 2x \cdot 3 + 3^2 = 4x^2 +$

$12x + 9$

9. *Factoring by common factor:* $2x^3 - 4x^2 + 6x^2y = 2x^2(x - 2 + 3y)$

9.1. Find the common factor: $2x^2$

9.1.1. Find the GCD of the coefficients: $\text{GCD}(2,4,6)=2$

9.1.2. Find the GCD of common variables: $\text{GCD}(x^3, x^2, x^2) = x^2$

9.2. Divide terms by the common factor:

$$\frac{2x^3}{2x^2} = x \text{ and } \frac{-4x^2}{2x^2} = -2 \text{ and } \frac{6x^2y}{2x^2} = 3y$$

10. *Factoring the quadratic form:*

$$x^2 + Sx + P = (x + a)(x + b), a \cdot b = P \text{ and } a + b = S:$$

$$x^2 + 5x + 6 = (x + 2)(x + 3)$$

- 10.1. Identify $P = a \cdot b$ and $S = a + b : P = 6$ and $S = 5$
- 10.2. Find the pairs of integers a, b that have a product of $P = 6$:
 $1 \cdot 6 = 6$ or $(-1) \cdot (-6) = 6$ or $2 \cdot 3 = 6$ or $(-2) \cdot (-3) = 6$
- 10.3. Find the pair a, b that gives a sum of $S = 5$: $2 + 3 = 5$
- 10.4. Write the factored form: $(x + a)(x + b) = (x + 2)(x + 3)$

The main reason for developing such a broad and deep domain expertise model was the investigation and confrontation of the *scaling-up problem*: despite the success of model-tracing tutors, in the majority of implementations, the tutor teaches a very elementary (low level) cognitive skill in isolation (Aleven, McLaren & Sewall 2009). However, even in school textbooks, medium difficulty exercises demand the application of a multitude of composite (top-level) cognitive skills in combination with each other. Their solutions demand the application of more high-level skills, like the identification and decomposition of the top level skills that appear in the exercise.

Table 2.1. Expanding $-10(x - 1)(x + 1)$ in three different ways

Operation	Result
A1. Monomial-polynomial multiplication	$-10(x - 1)(x + 1) =$ $(-10x + 10)(x + 1) =$
A2. Polynomial multiplication	$-10x^2 - 10x + 10x + 10 =$
A3. Collection of like terms	$-10x^2 + 10$
B1. Polynomial multiplication	$-10(x - 1)(x + 1) =$ $-10(x^2 + x - x - 1) =$
B2. Monomial-polynomial multiplication	$-10x^2 - 10x + 10x + 10 =$
B3. Collection of like terms	$-10x^2 + 10$

C1. Identity $(a+b)(a-b) = a^2 - b^2$	$-10(x-1)(x+1) =$
C2. Monomial-polynomial multiplication	$-10(x^2 - 1) =$
	$-10x^2 + 10$

To illustrate this situation, consider the algebraic expression $(x-3)^2 - 10(x-1)(x+1)$. In order to expand this expression, the student must first identify the operations that must be performed: a square of difference $(x-3)^2$, and a multiplication with three factors $-10(x-1)(x+1)$. Especially for the multiplication, the student can perform it in three different ways, described in Table 2.1.

So, it becomes clear that, even for a simple expansion exercise like the one in Table 2.1, a broad and deep domain expertise model containing all the potential skills is needed. In addition, intelligent recognition of the operations that are present in the expression is needed, whereas this is also a new cognitive skill that the tutor must be able to teach.

2.2.2 Intelligent Task Recognition

The key issue for tackling the scaling-up problem is the recognition by the tutor of the task(s) that must be performed, as well as of those entered by the student in order to match them, so as to provide guidance and feedback in each step of the tutoring process. In the MATHESIS tutor, these problems are tackled by parsing the MathML representation of the algebraic expressions and generating multiple internal representations. To illustrate how this is done, the algebraic expression $4x(x+7) + 48$ will be used:

1. The tutor gets a tree representation of the expression's MathML presentation, analogous to the Document Object Model (DOM) of HTML. This is provided by the Input Control applet (Figure 2.1) through JavaScript scripting. In this MathML DOM tree, every element of the algebraic expression is represented as a node.

MathML Presentation code for expression $4x * (x + 7) + 48$ without IDs	MathML Presentation code for expression $4x * (x + 7) + 48$ with IDs
<pre> <math> <mrow> <mrow> <mn>4</mn> <mi>x</mi> <mo>*</mo> <mo>(</mo> <mi>x</mi> <mo>+</mo> <mn>7</mn> <mo>)</mo> <mo>+</mo> <mn>48</mn> </mrow> </mrow> </math> </pre>	<pre> <math> <mrow id='1'> <mrow id='1.1'> <mn id='1.1.1'>4</mn> <mi id='1.1.2'>x</mi> <mo id='1.1.3'>*</mo> <mo id='1.1.4'>(</mo> <mi id='1.1.5'>x</mi> <mo id='1.1.6'>+</mo> <mn id='1.1.7'>7</mn> <mo id='1.1.8'>)</mo> <mo id='1.1.9'>+</mo> <mn id='1.1.10'>48</mn> </mrow> </mrow> </math> </pre>

Fig. 2.2. MathML Presentation code for expression $4x * (x + 7) + 48$ before and after intelligent task recognition

2. This MathML DOM tree is parsed using special methods provided by the Input Control. Each element of the expression (node) is given a unique identification string (id), which is used in the internal representations of the expression to uniquely identify each element (Figure 2.2). At the same time, the “atomic” elements such as numbers, variables and operation symbols are grouped in

mathematical objects like monomials and polynomials. These are represented using custom JavaScript objects, and they also get unique identification strings. For each monomial, its coefficient, variables and their exponents are kept along with their unique identification numbers. For each polynomial, its monomial terms are kept. In the case of expression $4x(x + 7) + 48$, four monomials are created, $4x$, x , 7 and 48 , as well as a polynomial, $x+7$, having as its terms the monomials x and 7 (.

ParsedMonomial	Monomial_1	Monomial_2	Monomial_3	Monomial_4
Coefficient	4	1	7	48
Variables	[x]	[x]	[]	[]
Exponents	[1]	[1]	[]	[]
idString	"1.1.1"	"1.1.5"	"1.1.7"	"1.1.10"
signID	""	""	"1.1.6"	"1.1.9"
coefficientID	"1.1.1"	""	"1.1.7"	"1.1.9"
variablesID	["1.1.2"]	["1.1.5"]	[]	[]
exponentsID	[""]	[""]	[]	[]

```

Polynomial_1 {
  monomials = [ Monomial_2, Monomial_3 ],
  exponent = 1
}

```

```

SumTerms {
  SumTerm_1 {
    Factors = [ Monomial_1, Polynomial_1 ]
  }
  SumTerm_2 {
    Factors = [ Monomial_4 ]
  }
}

```

```

allowedOperations = [ "1.1.3", "1.1.6" ]

```

```

allowedOperads = [ [Monomial_1, Polynomial_1 ], [ Monomial_2, Monomial_3 ] ]

```

Fig. 2.3 Mathematical objects created by intelligent task recognition for expression

$$4x * (x + 7) + 48$$

3. Identifying each operations' precedence is a key top-level skill for the expansion and factoring of algebraic expressions. As it will be explained in the next subsection, the tutor teaches students the correct order of operations. Consequently, the intelligent parsing mechanism extracts this information from the algebraic expression and represents it appropriately.
4. Finally, using the precedence of operations, the expression is represented as a sum of products using JavaScript arrays. The expression $4x(x + 7) + 48$ is represented as a sum array of two product arrays, $4x(x + 7)$ and 48. The first product array has two factors, monomial $4x$ and polynomial $(x + 7)$, while the second product array has only one monomial, 48.

All this information is extracted and represented for the expression to be rewritten (Figure 2.3). When the student selects a part (or the whole) of the expression, this part is parsed again and the same information is extracted and represented by the tutor; however, now the parser does not assign identification strings to the elements of the selected expression but just gets the ones assigned by the original parsing of the expression. As a result, the tutor can identify exactly which part of the expression is selected, which operations are selected and whether they have the right precedence to be performed. Moreover, when the student suggests what kind of operation he/she has selected, the tutor can check whether this suggestion is correct. For example, in expression $4x(x + 7) + 48$, if the student selects $4x(x + 7)$ and proposes "Common Factor", the tutor checks its internal representation and sees that the selected (sub)expression is not a sum and therefore it can't be factored. If the student selects the whole expression, the tutor sees that the expression is a sum with two terms and only then tries to extract a common factor. If it finds one, it proceeds by asking the student to give the common factor.

Otherwise, the student is given feedback that no common factor exists. Moreover, the tutor checks that the student has selected the whole expression, since there is no point in getting a common factor of part of an expression.

This approach, with exhaustive and multiple representations of the algebraic expressions allows the tutor to handle even more subtle conditions like dealing with the commutative properties of addition ($x + y = y + x$) and multiplication ($x \cdot y = y \cdot x$). In practice, the commutative property means that in a sum or product, the order of the terms is not important. By representing the algebraic expressions as a sum of products, the MATHESIS tutor can easily check student answers that are sums or products. Thus, when expanding the expression $(x + y^2)^2$, the tutor can accept as a correct answer any of the expressions $x^2 + 2xy^2 + y^4$, $y^4 + 2xy^2 + x^2$, $x^2 + 2y^2x + y^4$ and $x^2 + y^4 + 2y^2x$. Moreover, it can detect if a term is missing or is wrong and give the appropriate feedback. This performance is achieved by JavaScript functions that compare the sum and product arrays.

The overall result of this intelligent parsing is that the tutor can handle any algebraic expression that contains the math tasks (operations) described in the previous section. Therefore, the student can type any such expression and the MATHESIS tutor will parse it, detect which tasks are contained in it and guide the student appropriately. This feature is called *intelligent task recognition*. It is this feature combined with the broad and deep domain model that deals directly with the scaling-up problem: the MATHESIS tutor can handle *any* algebraic expression containing *any* combination of the math tasks described in the previous section. Thus, the MATHESIS tutor can guide a student in expanding expressions like $(2x + 3)^2 - 2(2x + 3)(2x - 3) + (2x - 3)^2$ or factor expressions like $4(x - 1)^2 - 9(x - 2)^2$.

2.2.3 The Tutoring Model: Deep Model Tracing With Intelligent Task Recognition

Equipped with such a detailed cognitive model, the MATHESIS tutor is able to exhibit expert human-like performance. The tutor makes all the cognitive tasks explicit to the student through the structure of the interface. The whole process is described below using as an example a real student interaction with the tutor for factoring the algebraic expression $4x * (x + 7) + 48$:

1. The student enters the algebraic expression in one of the ways described in Section 2.2.
2. The student starts the tutor by clicking “Start Exercise”, the tutor analyses the expression and *recognizes* the operations and their operands. As a result, the tutor displays an abstract representation of the algebraic expression, where each monomial in the expression has been substituted by an “*m*”. Thus, the algebraic expression $4x * (x + 7) + 48$ is represented as $m * (m + m) + m$ (Figure 2.1, Student Answering area). The purpose of this *intelligent task recognition* feature is to help the student understand the operations present in the expression through a visual, simplified and compact representation of the algebraic expression. It was realized that the use of letter “*m*” for representing a monomial could confuse the students, since this letter is normally used in mathematics to represent a variable. To avoid any such misconception, pen and paper exercises were given to the students, before using the system, where they had to transform algebraic expressions to the tutor’s “*m*” letter representation (this is a common practice followed by human tutors). After a few exercises, all students, even the weakest ones, were able to correctly perform this transformation. On the other hand, alternative representations were considered. For example, one of them was to use

- empty squares instead of “ m ”; however, it was abandoned as an option because a square symbol was used by the MATHESIS tutor to provide templates that guide student input (see step 4, below). Using a tree representation of the algebraic expression was also considered. However, in pen and paper exercises, where students were asked to transform between natural and tree representation, significant cognitive load and confusion were observed.
3. The student selects a part (or the whole) of the expression and then chooses from a drop-down list the operation that he/she believes corresponds to that part. In Figure 2.4 the student selected the whole expression $4x * (x + 7) + 48$ (highlighted) and the operation “FACTORING – Common Factor” from the drop-down list. It must be noted that this tutoring step is not part of the “traditional” tutoring practice in the Greek educational system and, to the best of the author’s knowledge, in many other educational systems. However, based on the author’s personal tutoring experience, this step is considered to be crucial and constitutes what is known in expert systems as an *expert’s blind spot*. Math teachers tend to believe that once students have been taught and practiced each operation separately, they are able to recognize and perform them when they appear in more complicated algebraic expressions. The author’s personal tutoring experience suggests that quite often students *don’t know what to do* because they cannot recognize which operations are present and the human tutor has to guide them in analyzing the expression under consideration. It is this step, in combination with the abstract representation of the algebraic expression presented in the previous step, that makes the analysis of the algebraic expression explicit to the student.

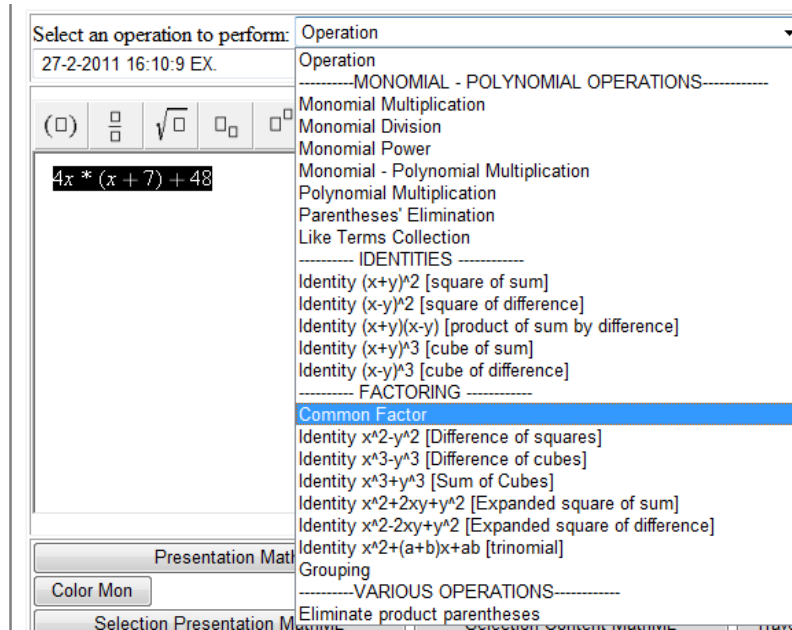


Fig. 2.4. The student proposes the operation “FACTORING-Common Factor” from the drop-down list of supported operations to be applied to the selected expression.

4. The tutor, based on the results of the intelligent task recognition (step 2), confirms and continues or informs the student that the suggested operation is not correct. In Figure 2.5, the suggested operation, “Common Factor”, is correct; the tutor confirms that with an appropriate message and starts guiding the student to perform the operation in a step-by-step manner (Figure 2.5, top, messages 2.1 and 2.2).

1.1 Select an exercise and click the 'Start Exercise' button
 1.2 Select a part of the algebraic expression and the operation that corresponds to that part
 2.1 Operation being executed: Factoring by Common Factor
 2.2 Please, enter the common factor. Write into the squares which don't have an exponent, the numbers with the correct sign

Select an operation to perform: Common Factor Exercise
 27-2-2011 16:10:9 EX ANSWERING SPACE Check Operation

(□) □ √□ □ □ □ ∞ θ < > ÷ × ↺ ? α γ

4x * (x + 7) + 48 □

Fig. 2.5. The tutor checks and confirms the student's suggested operation "Common Factor" through messages 2.1 and 2.2 (top). The common factor under question here is 4, denoted by the empty square scaffold in the "ANSWERING SPACE" area (bottom right).

The tutor also knows that the common factor for the expression $4x * (x + 7) + 48$ is the greatest common divisor of 4 and 48, that is, 4. The author's personal tutoring experience suggests that most students have considerable difficulties in finding the common factor. For this reason, the tutor displays in the student's answering area a visual scaffold of the common factor's form. Here, the common factor is only a number, denoted by a single square (Figure 2.5, bottom right). The tutor also displays a message that explains the meaning of the scaffold (Figure 2.5, top, message 2.2). It must be noted that the tutor supports two other kinds of common factors: variables with exponents, denoted as \square^\square and parentheses with exponents, denoted as $(\square)^\square$.

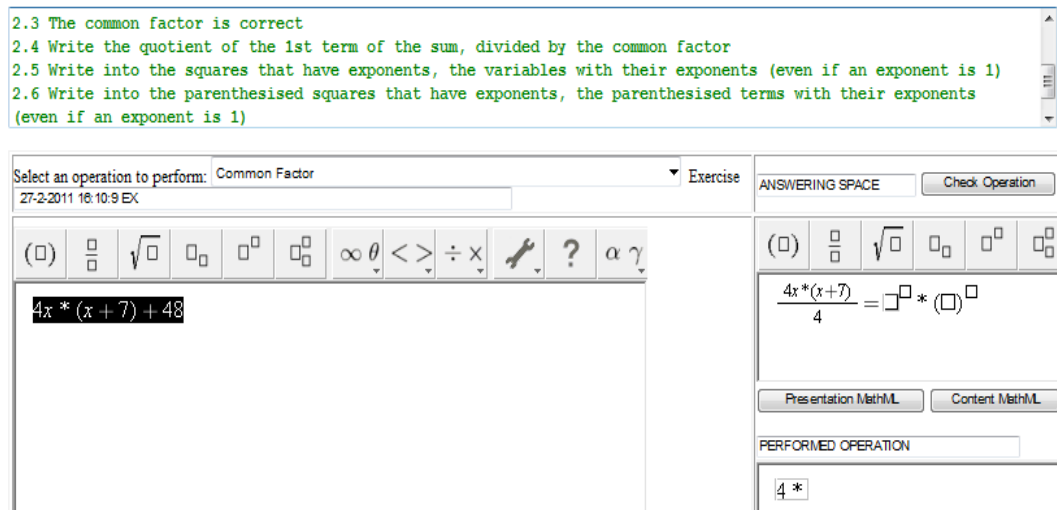


Fig. 2.6. The tutor confirms the entered common factor and asks for the first quotient by messages 2.3 and 2.4 (top). The quotient under question is $\frac{4x*(x+7)}{4} = x * (x + 7)$ denoted by the $\square * (\square)$ scaffold in the “ANSWERING SPACE” area (right).

5. The student correctly enters 4 in the position indicated “ANSWERING SPACE” as the common factor and clicks the “Check Operation” button. The tutor performs *intelligent parsing* on the student’s answer and confirms that it is correct (Figure 2.6, top, message 2.3). The tutor also displays the common factor followed by a multiplication symbol, $4*$, in the “PERFORMED OPERATION” area (Figure 2.6, bottom right). The purpose of this area is to display the steps that have been performed in multi-step math skills. Now, the student must divide each one of the terms of the sum, i.e. $4x * (x + 7)$ and 48, by the common factor. The first quotient that the student must calculate is $\frac{4x*(x+7)}{4} = x * (x + 7)$. The tutor displays the quotient and a visual scaffold of the expected answer in the “ANSWERING SPACE” area (Figure 2.6, right). The visual scaffold is $\square * (\square)$ denoting the expected answer $x^1 * (x + 7)^1$.

6. The student enters in the squares of the visual scaffold $\square * (\square)^\square$ the correct answer, $x^1 * (x + 7)^1$ and clicks the “Check Operation” button. Once again, the tutor performs *intelligent parsing* on the student’s answer and confirms that it is correct (Figure 2.7, top, message 2.7). The tutor also displays the expression $4 * (x * (x + 7))$ in the “PERFORMED OPERATION” area (Figure 2.7, bottom right) to denote the progress of the factoring process. The second quotient that the student must calculate is $\frac{48}{4} = 12$. The tutor displays the quotient and a visual scaffold of the expected answer in the “ANSWERING SPACE” area (Figure 2.7, right). The visual scaffold is \square denoting the expected answer 12.

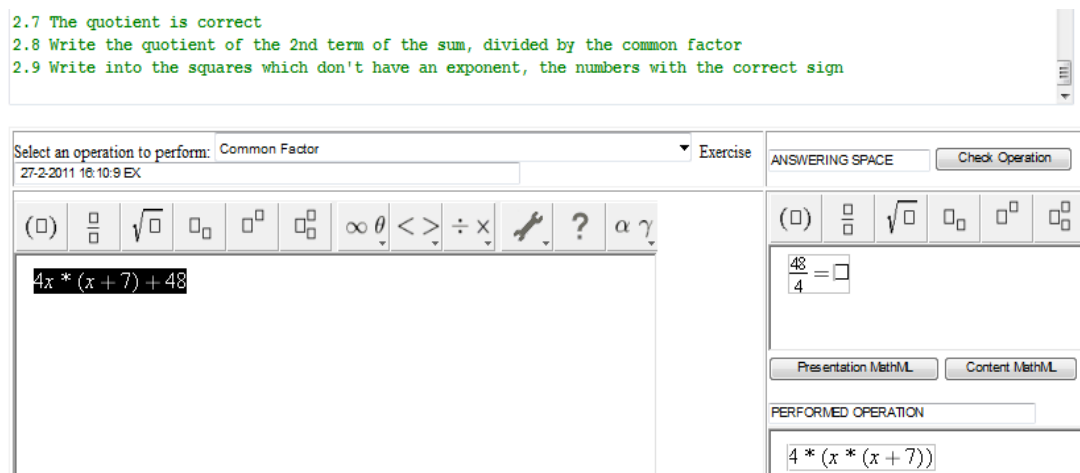


Fig. 2.7. The tutor confirms the first quotient and asks for the second quotient through messages 2.7 and 2.8 (top). The quotient under question is $\frac{48}{4} = 12$ denoted by the empty square scaffold in the “ANSWERING SPACE” area (right).

7. As soon as the student correctly enters the second quotient, the tutor displays a confirmation message (Figure 2.8, top, messages 2.10 and 2.11), rewrites the expression $4 * (x * (x + 7) + 12)$, parses the rewritten expression, displays its

abstract representation and prompts the student to perform the next operation, as shown in Figure 2.8.

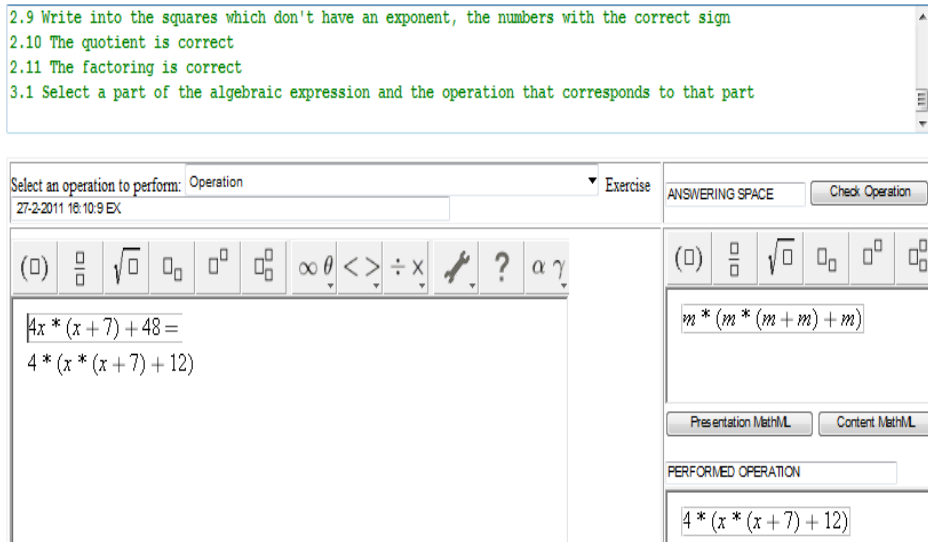


Fig. 2.8. Successful completion of the common factor method in expression $4x * (x + 7) + 48$.

- The student now selects $x * (x + 7)$ and performs monomial–polynomial multiplication. Once more the tutor exhibits its deep model tracing behavior and guides the student step-by-step to perform the two monomial multiplications, $x * x$ and $x * 7$ yielding $x^2 + 7x$. The result of this operation is shown in Figure 2.9.

The screenshot displays the MATHESIS Algebra Tutor interface. At the top, a scrollable text area contains the following instructions:

- 4.5 Multiply the monomial with the 2nd term of the polynomial
- 4.6 Write your answer in the ANSWERING SPACE and click the CHECK OPERATION button
- 4.7 The partial product is correct
- 4.8 The multiplication is correct
- 5.1 Select a part of the algebraic expression and the operation that corresponds to that part

Below the instructions is a control panel with a dropdown menu for "Operation" and a button for "Exercise". The main workspace is divided into two sections:

- Left Section:** Contains a toolbar with mathematical symbols (parentheses, fractions, square roots, exponents, infinity, less than, greater than, division, multiplication, undo, redo, question mark, alpha, gamma) and a text area with the following expressions:

$$4x * (x + 7) + 48 =$$

$$4 * (x * (x + 7) + 12) =$$

$$4 * (x^2 + 7x + 12)$$
- Right Section:** Labeled "ANSWERING SPACE", it contains a toolbar with the same mathematical symbols as the left section. Below the toolbar, the expression $m * (m + m + m)$ is entered. At the bottom, there are buttons for "Presentation MathML" and "Content MathML", and a section labeled "PERFORMED OPERATION" which displays $x^2 + 7x$.

Fig. 2.9. Successful completion of the monomial-polynomial multiplication $x * (x + 7)$.

- The student selects $x^2 + 7x + 12$ and performs factoring of the quadratic form $x^2 + Sx + P$ (trinomial). In order to achieve this, the student must find two integers a and b , such that $a \cdot b = P = +12$ and $a + b = S = +7$. The tutor, tracing its deep math domain model, guides the student in detail. First, the tutor prompts the student to identify $a \cdot b$ and $a + b$ (Figure 2.10, top, message 6.2) and displays the corresponding scaffold in the “ANSWERING SPACE” (Figure 2.10, right). The student correctly enters 12 and 7 for $a \cdot b$ and $a + b$ correspondingly (not shown in Figure 2.10).

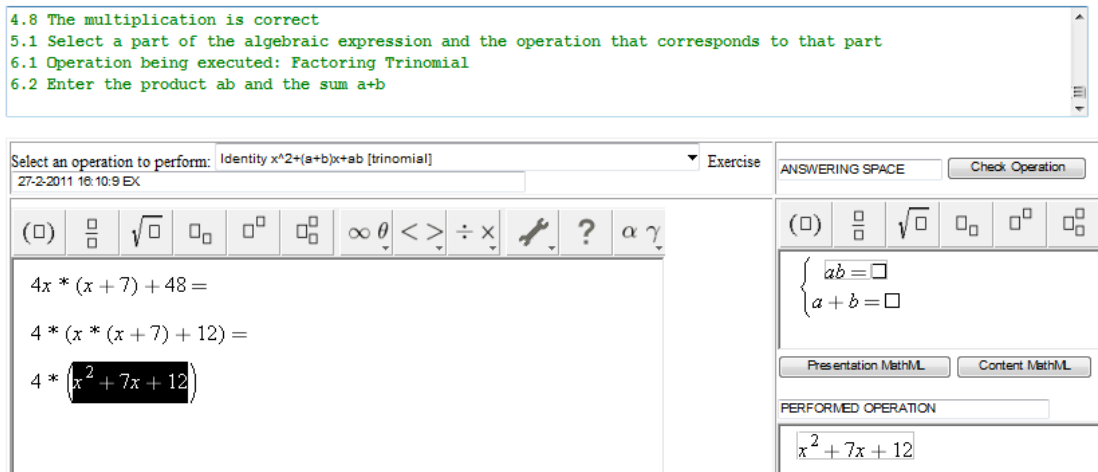


Fig. 2.10. First step of factoring $x^2 + 7x + 12$. The student must identify $a \cdot b = P = +12$ and $a + b = S = +7$.

10. The student now has to discover that $a=3$ and $b=4$. The student enters the incorrect answer $a=2$ and $b=6$ (this step is not shown). The tutor displays an error message and suggests the possible pairs of values for a and b (Figure 2.11, top, message 6.4), asking again for the values of a and b (Figure 2.11, right)). It must be noted that, for each one of the supported elementary skills, the model contains possible mistakes that the student might make. Each mistake is associated with error messages of varying depth, ranging from general suggestions down to the correct answer for the subtask. The depth and order of these messages are preset.

6.2 Enter the product ab and the sum $a+b$
 6.3 Enter a and b
 6.4 The numbers you have entered don't have a sum of 7. You must enter one of the pairs: 1 and 12, -1 and -12, 2 and 6, -2 and -6, 3 and 4, -3 and -4

Select an operation to perform: Identity $x^2+(a+b)x+ab$ [trinomial] Exercise
 27-2-2011 16:10:9 EX

() $\frac{\square}{\square}$ $\sqrt{\square}$ \square_{\square} \square^{\square} $\square^{\square}_{\square}$ $\infty \theta$ $<$ $>$ \div \times $\frac{\square}{\square}$ $?$ α γ

$4x * (x + 7) + 48 =$
 $4 * (x * (x + 7) + 12) =$
 $4 * (x^2 + 7x + 12)$

ANSWERING SPACE Check Operation

() $\frac{\square}{\square}$ $\sqrt{\square}$ \square_{\square} \square^{\square} $\square^{\square}_{\square}$

$\begin{cases} a = \\ b = \end{cases}$

Presentation MathML Content MathML

PERFORMED OPERATION

$x^2 + 7x + 12$

Fig. 2.11. Responding to a student error. The tutor displays an error message, gives help (top, message 6.4) and asks for the correct answer (right).

6.4 The numbers you have entered don't have a sum of 7. You must enter one of the pairs: 1 and 12, -1 and -12, 2 and 6, -2 and -6, 3 and 4, -3 and -4
 6.5 The entered values for a and b are correct
 7.1 Select a part of the algebraic expression and the operation that corresponds to that part

Select an operation to perform: Operation Exercise
 27-2-2011 16:10:9 EX

() $\frac{\square}{\square}$ $\sqrt{\square}$ \square_{\square} \square^{\square} $\square^{\square}_{\square}$ $\infty \theta$ $<$ $>$ \div \times $\frac{\square}{\square}$ $?$ α γ

$4x * (x + 7) + 48 =$
 $4 * (x * (x + 7) + 12) =$
 $4 * (x^2 + 7x + 12) =$
 $4 * ((x + 3) * (x + 4))$

ANSWERING SPACE Check Operation

() $\frac{\square}{\square}$ $\sqrt{\square}$ \square_{\square} \square^{\square} $\square^{\square}_{\square}$

$m * ((m + m) * (m + m))$

Presentation MathML Content MathML

PERFORMED OPERATION

$(x + 3) * (x + 4)$

Fig. 2.12. Successful completion of factoring $4x * (x + 7) + 48$.

11. The student now enters the correct answer, $a=3$ and $b=4$ (not shown). The tutor checks the answer, confirms and rewrites the expression, yielding $4 * ((x + 3) * (x + 4))$. The factoring of $4x * (x + 7) + 48$ is now successfully completed (Figure 2.12).

Once again, the scaling-up problem appears. The student could have followed a completely different solution path for factoring $4x * (x + 7) + 48$. The MATHESIS Algebra Tutor, based on its broad and deep expertise model as well as on the intelligent task recognition feature, is able to recognize this path and guide the student along.

Table 2.2 presents an alternative path in the solution space tree, involving only the top-level math skills (algebraic operations) the student could have followed and not the actual interaction with the tutor. As shown before, each one of these operations is a complex task that must be performed in a series of steps. The calculation of the quotient $\frac{4x*(x+7)}{4} = x * (x + 7)$ presented in step 5 (Figure 2.6) demanded the development of a model for calculating quotients of arbitrary complexity, like, e.g., $\frac{8x^2y^3(x+2)^2(2x-1)^3}{2xy^2(x+2)(2x-1)^2}$. Equally complex is the task of finding two integers with a given product and sum, like the task presented in steps 9-11. As a consequence, if someone tried to draw the solution space tree for the factoring of expression $4x * (x + 7) + 48$ it would end up with a tree of considerable breadth and depth. The *fine-grained modelling* of each top level math skill (algebraic operation) and its sub-skills in conjunction with the *intelligent task recognition* described in the previous section, allows the MATHESIS Algebra tutor to guide the student throughout this broad and deep solution space. Thus, this feature is called *deep model tracing*.

Table 2.2. Alternative Path for Factoring $4x * (x + 7) + 48$

Operation	Result
Initial expression	$4x * (x + 7) + 48 =$
1. Monomial-polynomial multiplication	$4x^2 + 28x + 48 =$
2. Common Factor	$4(x^2 + 7x + 12) =$
3. Factor $x^2 + Sx + P$	$4(x + 3)(x + 7)$

2.2.4 The Student Model

Based on the breadth and depth of its math domain expertise model, the tutor creates and maintains in a database a deep and broad student model. For every step of the student's attempted solution, the tutor records the following information:

- *Skill*: The algebraic operation that the student tried to perform in the specific step, e.g., “common factor calculation”.
- *Expression*: The algebraic expression on which the algebraic operation was performed, like $4x * (x + 7) + 48$.
- *Answer*: The answer given by the student, for example $4x$.
- *Correct*: It signifies whether the answer was right (1) or wrong (-1).
- *Timestamp*: The date and time the step was performed.

This information is presented in a table, with one row for each solution step. The table for factoring the expression $4x * (x + 7) + 48$ is shown in Table 2.3. Rows with dark background emphasize incorrect steps. Both students and their teachers can see this tabular representation of the student's solution steps.

Table 2.3. The Fine-Grained Student Model: Solution Steps

Skill	Expression	Answer	Correct
Automatic expression rewriting	$4x * (x + 7) + 48$	<i>(this step is performed by the tutor)</i>	1
Recognise the existence of a common factor	$4x * (x + 7) + 48$	Common factor	1
Calculate common factor	$4x * (x + 7) + 48$	4	1
Calculate the quotient of a term over the common factor	$\frac{4x * (x + 7)}{4}$	$x * (x + 7)$	1
Calculate the quotient of a term over the common factor	$\frac{48}{4}$	12	1
Automatic expression rewriting	$4x * (x + 7) + 48 = 4 * (x * (x + 7) + 12)$	<i>(this step is performed by the tutor)</i>	1
Recognise a monomial by polynomial multiplication	$x * (x + 7)$	monomial by polynomial multiplication	1
Monomial multiplication	$x * x$	x^2	1
Monomial multiplication	$x * 7$	$7x$	1
Monomial by polynomial multiplication	$x * (x + 7)$	$x^2 + 7x$	1
Automatic expression rewriting	$4 * (x * (x + 7) + 12) = 4 * (x^2 + 7x + 12)$	<i>(this step is performed by the tutor)</i>	1
Recognise trinomial	$x^2 + 7x + 12$	Trinomial	1
Identify a and b	$a \cdot b = 12, a + b = 7$	$\begin{cases} a = 2 \\ b = 6 \end{cases}$	-1
Identify a and b	$a \cdot b = 12, a + b = 7$	$\begin{cases} a = 3 \\ b = 4 \end{cases}$	1
Automatic expression rewriting	$4 * (x^2 + 7x + 12) = 4 * ((x + 3) * (x + 4))$	<i>(this step is performed by the tutor)</i>	1

In addition, the tutor can display statistics over a selected period of time about a specific cognitive skill, as shown in Figure 2.13. When a specific skill is selected, a table presenting the performance of the skill is displayed (Table 2.4).

It becomes obvious that such a detailed and time-stamped student model creates a digital timeline of the student's math skill mastery over time, with a number of possible uses: long term progress assessment, recent mastery status, automatic selection of exercises based on the student's weaknesses. The latter is not yet implemented in the system.

variables with no exponents are considered to have an exponent equal to one: 3/3 = 100%
do not introduce to a monomial result a variable that does not exist: 3/3 = 100%
divide numerical coefficients of monomials: 4/4 = 100%
divide main parts of monomials: 3/3 = 100%
divide all variables in main parts: 3/3 = 100%
subtract exponents of common variables: 3/4 = 75%
when a number or a variable has a zero exponent it is equal to 1 and it must be disregarded: 3/3 = 100%
Automatic expression rewriting: 2/2 = 100%
Recognise the existense of a common factor: 1/2 = 50%
Calculate common monomial factor: 1/2 = 50%
Calculate common polynomial factor: 1/1 = 100%
Calculate common factor: 2/4 = 50%
Calculate the quotient of a monomial term over the common factor: 3/4 = 75%
Calculate the quotient of a polynomial term over the common factor: 3/3 = 100%
Calculate the quotient of a sum term over the common factor: 5/7 = 71%
Select a skill

Fig. 2.13. The Student Model: Skill Performance Statistics

Table 2.4. Performance of skill “Calculate common factor”. The percentage is 2/4=50%

Operation	Exression	Answer	Correct	Date
Calculate common factor	$4x * (x + 7) + 48$	4	1	27-02-2011 16:55:33
Calculate common factor	$4x * (x + 7) + 48$	4	1	22-02-2011 18:26:07
Calculate common factor	$4x * (x + 7) + 48$	$4x$	-1	22-02-2011 18:26:02
Calculate common factor	$4x * (x + 7) + 48$	$4x$	-1	22-02-2011 18:19:53

2.3 THE LEARNING MANAGEMENT SYSTEM

The MATHESIS Intelligent Algebra Tutoring School is accessible through a web interface⁷. Each user gets a unique Username and Password. Users can register either as teachers or students. Students are guided to the MATHESIS Algebra Tutor interface (Figure 2.1), where they solve their assigned exercises as it was described in Section 2.2.3. Teachers are taken to the Teacher Menu (Figure 2.14), which provides links for the following managerial tasks:

⁷ <http://users.sch.gr/dsklavakis>

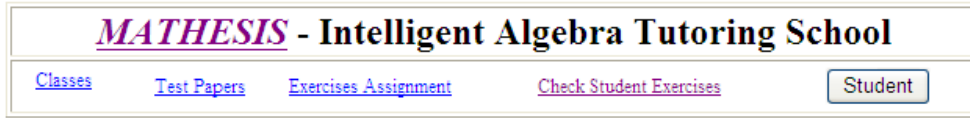


Fig. 2.14. The Teachers' Menu

- *Classes*: Teachers can create classes. For each class the teacher enters the real school, grade and name of the class. Students are registered to the class by their Usernames. That means that the students must be already registered in the system. Students can also be deleted from a class. (Figure 2.15).

MATHESIS - Classes Management

School	Grade	Class
TEST SCHOOL	C	C2

SN	Username	Fullname
<input type="checkbox"/>	1 daphne	DAPHNE
<input type="checkbox"/>	2 paris	PARIS
<input type="checkbox"/>	3 studentA	STUDENT_A

with Username:

Fig. 2.15. The Classes Management Page.

- *Test Papers*: The system provides an online HTML editor for the creation and editing of test papers (Figure 2.16). For each test paper the teacher enters the type of school, grade, book, chapter and section of a textbook that the contained exercises correspond to. Each test paper is also characterized as *public* or *private* (Figure 2.16a). Public test papers can be accessed and used (but not modified)

from any teacher registered in the system, while private ones can be used and edited only by their creator. Test papers are used for the assignment of exercises to students. Currently, the system provides five public test papers that contain exercises from the official textbook that is taught in the 3rd grade of Gymnasium (junior high school) in secondary education in Greece. Each test paper is an HTML page. Conceptually, each paper is organized as a set of exercises containing one or more questions. For each exercise, its questions are laid out in rows and columns using HTML tables. The author inserts new exercises by defining how many questions they contain and in how many rows and columns they will be arranged, using the “Insert Exercise” button and the corresponding fields (Figure 2.16, left, below the editor). The system creates the appropriate HTML code for the table and displays it in the editing area. It also generates check boxes with unique identification strings in front of the exercise and each of its questions (Figure 2.16b). These check boxes are used later for selecting and assigning exercises (Figure 2.17). The author adds any text for describing the exercise and its questions. In Figure 2.16, exercise 22 has just been added, containing 3 questions, arranged in one row and three columns, labeled by the author as ‘a)’, ‘b)’ and ‘c)’ (Figure 2.16b). Finally, for each question, the author enters the algebraic expression using a WebEq Input Control. In Figure 2.16, the author has just entered the expression $4x * (x + 7) + 48$ in question (a) of exercise 22 (Figure 2.16c). The system displays on the right side of the editor the test paper as an HTML page, using the MathML viewer MathPlayer to display properly the mathematical expressions (Figure 2.16d). The HTML code of each test paper is saved in a database, together with the papers’ information, and can be recalled and edited any time by changing, adding or deleting exercises. It must be

noted that, due to the intelligent task recognition feature of the tutor, the authors do not have to annotate or describe any solution steps for the questions.

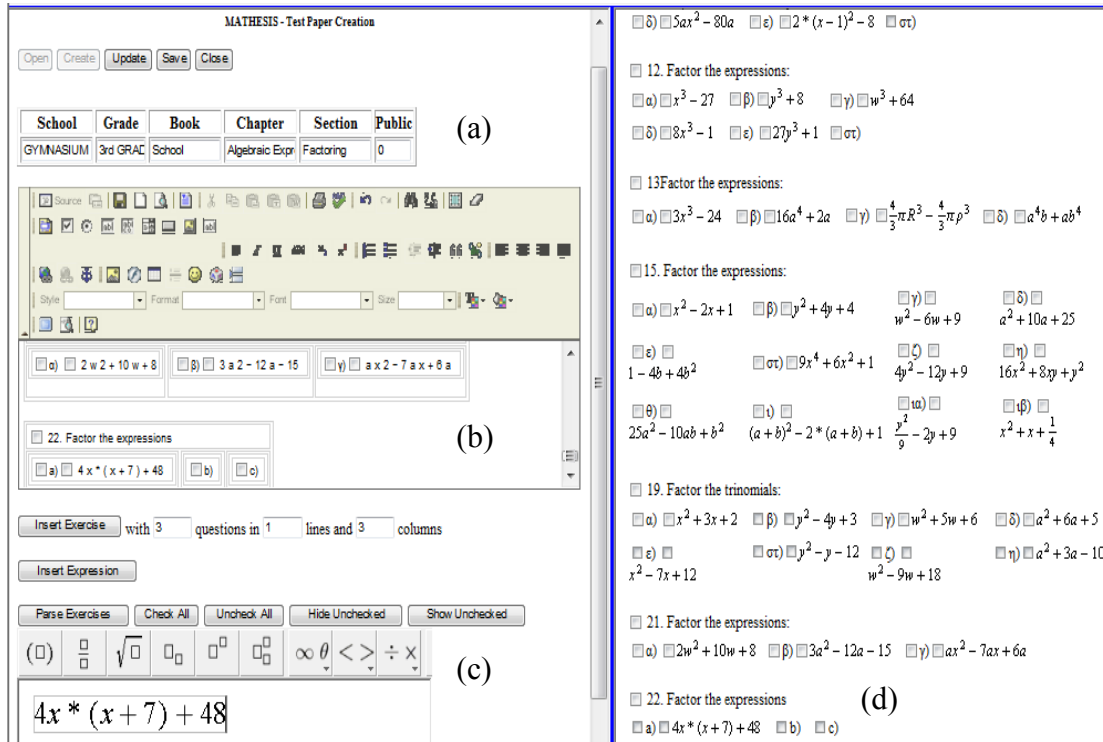


Fig. 2.16. Test Paper Editing. The author has just created exercise no. 22 using the HTML editor (b) and inserted expression $4x * (x + 7) + 48$ for the first question using the math editor (c). The paper is shown on the right with the newly added exercise at the bottom (d).

- *Exercise assignment:* The system provides tools for individualized assignment of exercises. The teacher can assign different exercises to different students, according to their performance. The assignment process is simple: The teacher selects a class and any student(s) from this class as well as a test paper and any exercise(s) from it. By checking the appropriate boxes, the selected exercise(s) are assigned to the selected student(s) (Figure 2.17).

MATHESIS - Exercises Assignment

Date: 1-3-2011

Test Paper: Exercises:

School	Grade	Book	Chapter	Section	Public
GYMNASIUM	3rd GRAC	School	Algebraic Exp	Factoring	0

Class: Students:

School	Grade	Class
TEST SCHOOL	C	C2

	SN	Username	Fullname
<input type="checkbox"/>	1	daphne	DAPHNE
<input checked="" type="checkbox"/>	2	paris	PARIS
<input checked="" type="checkbox"/>	3	studentA	STUDENT_A

1. Factor the expressions:

a) $3a + 6b$ b) $2x - 8$ c) $8w^2 + 6w$

d) $-9x^2 - 6x$ e) $8a^2b + 4ab^2$ f) $2x^2 - 2xy + 2x$

g) $a^2b + ab^2 - ab$ h) $2a^3 - 4a^2 + 6a^2b$ i) $\sqrt{2}xy - \sqrt{18}y + \sqrt{8}y^2$

2. Factor the expressions:

a) $x * (a - b) + y * (a - b)$ b) $a * (x + y) + b * (x + y)$ c) $(3x - 1) * (x - 2) - (x + 4) * (x - 2)$

d) $a^2 * (a - 2) - 3 * (2 - a)$ e) $4x * (x - 1) - x + 1$ f) $2x^2 * (x - 3) - 6x * (x - 3)^2$

3 i) Factor the expressions:

a) $x^2 + x$ b) $2y^2 - 5y$ c) $w * (w - 3) - 2 * (3 - w)$ d) $a * (3a + 1) - 4a$

3ii) Solve the equations:

a) $x^2 + x = 0$ b) $2y^2 = 5y$ c) $w * (w - 3) - 2 * (3 - w) = 0$ d) $a * (3a + 1) = 4a$

4. Factor the expressions:

a) $x^2 + xy + ax + ay$ b) $x^3 - x^2 + x - 1$ c) $x^3 - 5x^2 + 4x - 20$

d) $2x^3 - 3x^2 + 4x - 6$ e) $4x^2 - 8x - ax + 2a$ f) $9ab - 18b^2 + 10b - 5a$

Fig. 2.17. Individualized Assignment of Exercises to Students.

- Student assessment:* The solution steps taken by a student are recorded in the database and statistics are computed about the correct/incorrect performance of operations. These steps and statistics can be retrieved and viewed by the teacher. On the left side of Figure 2.18, the teacher selects the time interval for which he/she wants to assess the student(s). He/She opens a classroom and selects a student. The system displays in a drop down list all the test papers containing exercises that were assigned to the student during the selected time period. The teacher selects a test paper and its contents are displayed (Figure 2.18, right). Assigned exercises for which no solution was attempted by the student are marked in a red background. In Figure 2.18, these are questions 13a) $3x^3 - 24$ and 15a) $x^2 - 2x + 1$ located in the middle of the test paper (red color appears as dark grey in grayscale). Those with at least one attempted solution, either correct

or wrong, are marked with green background. In Figure 2.18 this is question 21a) $2w^2 + 10w + 8$ (green color appears as light grey in grayscale).

The screenshot shows a sidebar on the left with filters for 'Exercises', 'Classes', 'Students', 'Test Paper', and 'Statistics'. The main area displays a list of math problems. Question 21a) is highlighted in green, and questions 13a) and 15a) are highlighted in red. Callout boxes explain the highlighting: 'Questions 13a and 15a with no attempted solution marked in red' and 'Question 21a with an attempted solution marked in green'.

Fig. 2.18. Student Assessment: Selecting a Solved Exercise

By selecting an exercise and clicking the “Select Exercise” button, the attempted solution steps are displayed as shown in Table 2.3. The teacher can also select a specific math skill from the drop-down list on the lower left part of Figure 2.18. As mentioned in Section 2.2.3, the list displays all skills performed by the student with their corresponding percentage of correct performances during the selected time period, as shown in Figure 2.13. By selecting a specific skill, a table of the skill performances taken into account is displayed (Table 2.4).

2.4 RELATED WORK

The development of the domain expertise and the implementation of the model-tracing tutoring model in model-tracing tutors are so demanding in time and human resources (Aleven et al. 2006) that these tutors are currently developed by specialised research teams, they are usually experimental prototypes and they are used in strictly controlled and supervised educational settings, mainly in universities (VanLehn 2006). The most successful and widely used math MTTs are Cognitive Tutors developed by Carnegie Learning⁸, based on more than twenty years of cognitive science research at CMU (Koedinger & Corbett 2006). Cognitive Tutors are now an integral part of complete curricula used in hundreds of middle and high schools throughout the United States. However, despite their innovative nature and practical success, Cognitive Tutors are commercial products that have to adapt to very strict guidelines and educational goals of the US educational system. They have to follow the textbook by teaching specific exercises that train the students in specific cognitive skills. In the case of algebraic expressions' operations, they teach each operation separately and not in combinations with each other. They also teach a fixed set of exercises where all the anticipated solution steps are *pre-computed* by solving the problem in all acceptable ways by running a rule-based problem-solver (Van Lehn 2006). Therefore, these tutors do not tackle the problem of parsing an *arbitrary* algebraic expression, identifying the existence of any possible combination of operations and their precedence and following the student in any possible

⁸ www.carnegielearning.com

correct path of the solution space tree. In other words, they are not designed to deal with the scaling-up problem (Alevén, McLaren, & Sewall, 2009).

Another kind of pseudo-MTTs is the example-tracing tutors (Alevén, McLaren, Sewall, & Koedinger, 2009) under development at Carnegie Mellon University. There are two websites that provide example-tracing tutors for middle-school mathematics: the Mathtutor⁹ website (Alevén, McLaren, & Sewall, 2009) and the Assistments¹⁰ website (Razzaq, Feng, Nuzzo-Jones, Heffernan, Koedinger, Junker et al. 2005). Example-tracing tutors have a very narrow and shallow, exercise-specific, domain expertise model. They offer considerable reduction in development time but are even further away from dealing with the scaling-up issue.

*ActiveMath*⁷ is another web-based intelligent tutoring system for mathematics (Melis, Andrès, Büdenbender, Frischauf, Gogvadze, Libbrecht et al. 2001). The system aims mainly for adaptive guidance and presentation of mathematical content based on ontological representation of mathematical concepts, learning goals and acquired knowledge. However, when it comes to problem-solving skills, ActiveMath offers mainly multiple choice questions and some more interactive exercises. In these, the system does not guide the student along a solution path. It uses the external Computer Algebra Systems (CAS) to simply check the correctness of the student's solution. Therefore, the system completely avoids the hard problems of model tracing, that is, generating the correct solution(s) at each step, comparing the students' input, recognising errors and providing feedback.

⁹ <https://mathtutor.web.cmu.edu/>

¹⁰ www.assistments.org

⁷ www.activemath.org

⁸ www.aplusix.com

*Aplusix*⁸ is an Algebra Learning Assistant. After several years of research (Nicaud, Bouhineau, & Chaachoua, 2004), it is now a commercial product. It covers the domains of arithmetic calculations, expansion, simplification and factoring of algebraic expressions, solution of polynomial and rational equalities and inequalities. The system combines features of microworlds and Computer Algebra Systems. The student can type an algebraic expression, suggest its domain (calculation, expansion-simplification, factoring, solution) and enter the solution steps. At each step, the system checks the student's input for *equivalence* using encoded transformation rules. As a result of this type of checking, the system only suggests if the expression entered by the student is correct or incorrect, without any further feedback about the error committed. However, the student can ask for suggestions about the possible operations that he/she can perform and can also ask the system to perform them. We could say that the resulting tutoring model is almost equivalent with that of the MATHESIS tutor though less fine-grained. In unusual situations, this can lead the Aplusix system to “miss” intermediate student errors. For example, the expression $2x - (6 - x) - (6 + x)$ is correctly expanded and simplified by changing the signs of the parenthesized terms as in $2x - 6 + x - 6 - x = 2x - 12$. However, a student can arrive at the correct result by making the same mistake twice, that is, not changing the signs of $-x$ in the first parenthesis and of $+x$ in the second one, as in $2x - 6 - \underline{x} - 6 + \underline{x} = 2x - 12$! Moreover, the Aplusix system has considerable limitations to the kind of expressions that it can factor: polynomial expressions in one variable and degree no higher than 4, or in two variables and degree at most 2. It cannot handle expressions like $3x^2y^2z^3 - 6xyz^2$, $x^4 - y^4$, $(x + y)^2 - z^2$ or $4x(x + 7) + 48$.

As far as the author knows, the MATHESIS Algebra Tutor is unique with regard to the combined breadth and depth of its domain expertise model as well as the intelligent task recognition feature.

2.5 EVALUATION OF THE MATHESIS SYSTEM

The MATHESIS Algebra Tutor is a research prototype, performance-oriented, domain expert system with emphasis on the scaling up problem. The tutor is part of the MATHESIS project, which aims at the development of authoring tools for real world model-tracing math tutors. Therefore, the MATHESIS Algebra Tutor and the MATHESIS tutoring school built around it were designed to become part of real educational settings. For this reason, the following factors were taken into consideration:

- 1) *Teaching performance*: In order for an intelligent system to be used by teachers and students, it should contribute to observable positive learning outcomes. Besides any kind of scientific evaluation, teachers and students must feel and see that using the system helps students learn more effectively. It has been shown that model-tracing tutors do produce considerable learning outcomes, mainly because of their domain expertise models (Corbett 2001; Ritter, Kulikowich, Lei, McGuire, & Morgan 2007). In this work a holistic approach was adopted: developing a deep model of a sufficiently broad domain in mathematics with intelligent task recognition and deep model-tracing.
- 2) *Usability*: This factor is multidimensional, with the most important dimensions being:
 - a) Easy to learn and use interface. Care has been taken to keep the user interface as simple as possible – given the complex task of teaching that this interface must perform – and as close as possible to the “traditional” way of doing things. For the

- teachers, this means following the day-to-day workflow of selecting, assigning and assessing exercises. For the students, effort has been made to keep the problem-solving procedure as close as possible to the pen and paper paradigm without losing the benefits of a digital environment.
- b) Easy access to the system. The MATHESIS system is web based and therefore accessible anytime from anywhere, provided there is an internet connection. In addition, it has minimal requirements in hardware and connection speed.
- 3) *Scalability*: The set of exercises that the tutor is able to teach has to be of considerable breadth and depth. Limiting the set of supported exercises is a major factor of system rejection by the teachers. Teachers must be given the flexibility to choose exercises of different complexity and difficulty levels in order to accommodate the varying levels of competence of their students. The systems' deep and broad domain expertise model in conjunction with the intelligent task recognition system covers a considerable set of exercises.

2.5.1 Evaluation by Teachers

The system has been demonstrated to real math teachers, both through on-site live presentations and through invitations to use it online. The most extensive evaluation of the system was held in a three hours' workshop at the 2nd PanHellenic Conference on Digital and Web Applications in Education, held in Naoussa in April of 2010 (<http://hmathia10.ekped.gr/>) The purpose of the workshop was to teach math teachers the use of the system and investigate their attitude towards adopting the system in their everyday teaching. More specifically, the author wanted to investigate their opinions

regarding the following system features, which we consider the most decisive for the adoption of the system by a broad group of math teachers:

- The usability of the system.
- The ability to create their own exercises and assign them to individual students.
- The teaching performance of the system, particularly the depth and granularity of the domain model.
- The value of the fine-grained student model for their assessment tasks.

Forty (40) math teachers in secondary education participated in the workshop. Most of them were young, around 30 years old, self-motivated and positive in using computer programs for math teaching.

First, the teachers used the LMS to sign up, create students and enroll them to classes. Then, they used the existing test papers to assign exercises to their students. They have actually assigned one exercise for each one of the 16 top-level skills covered by the tutor as well as a few exercises with combinations of these skills. The teachers spent most of their time solving the assigned exercises as if they were students. They were also instructed to make deliberate mistakes to test the system's responses. They were also instructed to inspect the student model between the solutions of the exercises to see how this model was dynamically updated by their performance as students.

After using the system, the teachers filled in a short questionnaire. The questions and the teachers' answers are shown in Table 2.5. These questions are in direct correspondence with the aforementioned system features the author wanted to evaluate.

Thirty five teachers (87.5%) found the system easy or fairly easy to use (Question 1). Thirty two teachers (80%) agreed that it naturally follows the short- and long-term tutoring tasks workflow (Question 2). Twenty eight teachers (70%) appreciated the

freedom provided by the system to create their own work papers with their own exercises, as well as the ability for *individualized* assignment of exercises (Questions 3 and 4). Thirty two teachers (80%) found the fine grained student model unique and decisive when it came to assessment. However, five teachers (12.5%) considered that it might be too fine-grained for well-performing students. Three teachers (7.5%) complained that this step-by-step guidance of the model-tracing algorithm could be too authoritative and restrictive in the development of the students' self-confidence (Question 5). All forty (40) teachers were impressed by the human-like step-by-step guidance given to the student by the system and the ability to see the students' solution steps (Questions 6 and 7).

Table 2.5. Evaluation results given by forty (40) math teachers after a three-hour hands-on workshop (questions are translated from Greek)

Questions	Answers			
1. You find the overall use of the system...	Easy 31/40 (77.5%)	Fairly Easy 4/40 (10.0%)	Fairly Hard 3/40 (7.5%)	Hard 2/40 (5.0%)
2. How well does the Learning Management System fits your day-to-day teaching tasks?	Very much 19/40 (47.5%)	Much 13/40 (32,5%)	Quite well 8/40 (20.0%)	Not at all 0/40 (0.0%)
3. You find the ability to create your own exercises as...	Very Important 18/40 (45.0%)	Important 10/40 (25.0%)	Indifferent 12/40 (30.0%)	Useless 0/40 (0.0%)
4. You find the ability to assign different exercises to different students as...	Very Important 18/40 (45.0%)	Important 10/40 (25.0%)	Indifferent 12/40 (30.0%)	Useless 0/40 (0.0%)
5. Do you think that the level of analysis for the solution steps proposed for each operation is...	Excessive 8/40 (20.0%)	Normal 32/40 (80.0%)	Inadequate 0/40 (0.0%)	
6. How would you characterize the step-by-step guidance of the student?	Very Important 40/40 (100.0%)	Important 0/40 (0.0%)	Indifferent 0/40 (0.0%)	Useless 0/40 (0.0%)
7. How would you characterize	Very Important	Important	Indifferent	Useless

the ability to see the students' solution steps regarding his/her assessment?	40/40 (100.0%)	0/40 (0.0%)	0/40 (0.0%)	0/40 (0.0%)
---	-------------------	----------------	----------------	----------------

2.5.2 Evaluation in a Real Classroom

In late 2011 the system was also used and evaluated for three months in a third grade class (ages 14-15) of 20 students in a junior high school at the town of Drama, in northern Greece. The purpose of this evaluation was to integrate the use of the system in the normal, daily, official educational practice and investigate the following features:

- The usability of the system.
- The students' attitude towards the tutoring performance of the system, particularly the fine-grained, step-by-step guidance provided by the system.
- The affective impact of the system to the students, particularly the impact on frustration and fear during the solution of exercises.
- The potential raise of student performance.

Mathematics in this grade is taught four hours a week using the textbook, blackboard lessons and worksheet practice both in classroom and at home. In this evaluation three hours were taught in the traditional way using blackboard lessons and worksheet practice. The fourth hour was taught in the school's computer laboratory, where students used the MATHESIS system. Some of the students also used the system from their homes for extra practice. The system was evaluated by the students for its usability and tutoring behavior using short questionnaires (Table 2.6). The results of the students' evaluation are:

Usability: 85% of the students found the system easy to learn and use, while the rest 15% found it fairly easy to learn (Question 1). In practice, the first group of students

(85%) needed one or two 45-minute sessions with the system to get fully acquainted while the second group (15%) needed three or four sessions.

Tutoring performance: 75% of the students said that the guidance and assistance they got from the system was similar to the human tutor’s teaching. The rest 25% found the help and guidance of the system too detailed and fine grained (Question 2). These students were the best performing ones and they proposed that the system should allow the student to skip some “trivial” problem solving steps.

Affective impact: 85% of the students replied that the use of the system helped them to overcome the most common emotional problems they face with mathematics, that is, frustration and disappointment (Question 3). The reasons are that they have the time they need to think (75%), they get step-by-step guidance (65%), they have the freedom to try the solution steps they think correct (65%) and make mistakes (90%) (Question 4).

Table 2.6. Evaluation results given by twenty (20) students after a three-month period (questions are translated from Greek)

Questions	Answers			
1. You find the overall use of the system...	Easy 17/20 (85.0%)	Fairly Easy 5/20 (15.0%)	Fairly Hard 0/20 (0.0%)	Hard 0/20 (0.0%)
2. How would you characterize the step-by-step guidance of the tutor?	Too detailed 5/20 (25.0%)	Natural 15/20 (75.0%)	Inadequate 0/20 (0.0%)	
3. You find that your frustration when you solve an exercise with the tutor is...	Bigger 2/20 (10.0%)	Equal 1/20 (5.0%)	Lower 17/20 (85.0%)	
4. Which do you think are the most important advantages for you when using the tutor? (multiple answers)	Adequate time to think 15/20 (75.0%)	Freedom to make mistakes 18/20 (90.0%)	Step-by-step guidance 13/20 (65.0%)	Ability to try possible solutions 16/20 (65.0%)

Cognitive performance: It is the author’s belief that the most important attribute of an intelligent tutoring system is its cognitive performance, that is, its ability to build deep, long-term and transferable knowledge within the student’s minds. The cognitive performance of the MATHESIS Algebra Tutor was specifically tested in the domain of factoring, using the methods of *common factor and identities difference of squares* $x^2 - y^2 = (x + y)(x - y)$, *square of sum* $x^2 + 2xy + y^2 = (x + y)^2$ and *square of difference* $x^2 - 2xy + y^2 = (x - y)^2$. The students were initially taught this subject for six weeks without using the system at all. After this period, the students completed a test to assess mastery of the subject. Then, the students used the MATHESIS system for two weeks to solve all the relevant exercises provided by the system. Some of these exercises can be found in: Figure 2.17, exercises 1, 2, 3 and 4; Figure 2.16, exercise 15; and Figure 2.18, exercise 9. Right after they had completed these exercises, they took a post-test with exercises similar to those of the pre-test. The results are shown in Table 2.7. There, the pre-test items are denoted by “Pre”, while post-test items are denoted by “Post”. In the left column four pairs of exercises are shown. For each pair the pre-test and the post-test exercises are shown. The next three columns show the elementary math skills needed to correctly perform each factoring method. For each skill, the percentages of students who performed it correctly are shown both for the pre-test and post-test exercises.

Table 2.7. Students' performance rise by the MATHESIS Algebra Tutor

<p>Exercise 1</p> <p>Pre: $6xy^2 + 3x^2y - 9xy = 3xy(2y + x - 3)$</p> <p>Post: $8x^2y^3 - 4x^2y^2 - 12x^3y^4 = 4x^2y^2(2y - 1 - 3xy^2)$</p>	Math Skills					
	Recognize Common Factor Method		Calculate Common Factor		Calculate Quotients inside the parenthesis	
	Pre	Post	Pre	Post	Pre	Post
	85%	90%	65%	85%	70%	80%
<p>Exercise 2</p> <p>Pre: $4y^2 - 81 = (2y)^2 - 9^2 = (2y - 9)(2y + 9)$</p> <p>Post: $9 - 25x^2 = 3^2 - (5x)^2 = (3 - 5x)(3 + 5x)$</p>	Math Skills					
	Recognize Difference of Squares Method $x^2 - y^2 = (x + y)(x - y)$		Find the Squares		Apply the Identity	
	Pre	Post	Pre	Post	Pre	Post
	85%	95%	50%	65%	60%	80%
<p>Exercise 3</p> <p>Pre: $x^2 + 6xy + 9y^2 = x^2 + 2 \cdot x \cdot 3y + (3y)^2 = (x + 3y)^2$</p> <p>Post: $25a^2 - 60ab + 36b^2 = (5a)^2 - 2 \cdot 5a \cdot 6b + (6b)^2 = (5a - 6b)^2$</p>	Math Skills					
	Recognize Square of Sum Method $x^2 \pm 2xy + y^2 = (x \pm y)^2$		Find the Squares and the Double Product		Apply the Identity	
	Pre	Post	Pre	Post	Pre	Post
	85%	95%	50%	65%	60%	80%
<p>Exercise 4</p> <p>Pre: $a^4 - 81 = (a^2)^2 - 9^2 = (a^2 + 9)(a^2 - 9) = (a^2 + 9)(a + 3)(a - 3)$</p> <p>Post: $16 - x^4 = 4^2 - (x^2)^2 = (4 + x^2) \cdot (4 - x^2) = (4 + x^2) \cdot (2^2 - x^2) = (4 + x^2) \cdot (2 + x) \cdot (2 - x)$</p>	Math Skills					
	Recognize Difference of Squares Method $x^2 - y^2 = (x + y)(x - y)$		Find the Squares		Apply the Identity	
	Pre	Post	Pre	Post	Pre	Post
	9/20 45%	12/20 60%	7/20 35%	11/20 55%	7/20 35%	9/20 45%
	4/20 20%	9/20 45%	4/20 20%	6/20 30%	3/20 15%	6/20 30%
	(4/9 44%)	(9/12 75%)	(4/7 57%)	(6/11 55%)	(3/7 43%)	(6/9 67%)

Exercise 1 is a common factor method. Exercises 2 and 3 correspond to the three different identities mentioned above. Although they seem to share some identical sub-skills, like the “Find squares” and “Apply identity”, in practice the identity $x^2 \pm 2xy + y^2 = (x \pm y)^2$ is more demanding: the student has to verify that the third term is actually the double product of the two squares and take into account the sign of the double product. The similar success percentages in Exercises 2 and 3 do not reflect these subtle differences in the application of these identities. Exercise 4 is a more complex one. First, the term x^4 is a square of a square that is, $(x^2)^2$. Second, after the first application of the identity $x^2 - y^2 = (x + y)(x - y)$, the term $(4 - x^2)$, which is also a difference of squares, appears. These two difficulty factors significantly reduce the success percentages. In the pre-test only nine students (45%) recognized that $x^4 = (x^2)^2$ and of these students, only four (20%) factored the term $(4 - x^2)$. The corresponding results for the post-test (60% and 45% correspondingly) are considerably raised but still remain low.

It is the author’s opinion that this comparison further supports the empirical observation that in mathematics there are non-intuitive practical differences in what are formally “identical tasks”. It seems that the application of the same task (square recognition) in a more complicated expression, like x^4 , demands the recall and application of “deeper” sub-skills like the one expressed by the formula $x^{2n} = (x^n)^2$. In turn, this fact supports the necessity for broader and deeper models in intelligent tutoring systems. In any case, the results in Table 2.7 show a considerable performance rise, given the limited time of two weeks that the students had in their disposal for using the MATHESIS system.

2.6 DISCUSSION AND FURTHER WORK

The MATHESIS system and especially the MATHESIS Algebra Tutor is a successful proof-of-concept system. The basic research hypothesis of the MATHESIS project is that, in order to build successful intelligent real-world tutoring systems, we must build powerful domain expertise models. The engineering of such broad and deep models has to overcome the common obstacle of all expert systems, the *knowledge acquisition bottleneck*: the extraction of the expertise from domain experts and its representation in efficient ways. In the domain of knowledge engineering, the most profitable solution up to now is *knowledge reuse*, which is achieved through open, modular, interchangeable, inspect-able, formal knowledge representations and system implementations (Aitken & Sklavakis 1999). Equally important, the models must be deep and broad, having a wide basis of low level knowledge about simple task performance, on top of which is built the knowledge for performing higher level domain tasks. Otherwise, models are *brittle* (Lenat & Guha 1990), performance is limited, scaling up is intractable and the systems fail to cope with real-world demands. The author believes that the MATHESIS Algebra Tutor incorporates all these characteristics that make it a successful real-world intelligent tutoring system.

Of course, the system is an experimental prototype and more evaluation is needed. The teachers that took part in its evaluation were self-motivated and enthusiastic about the use of technology in education. Also, they did not use the system for a long period of time in their everyday teaching duties and they were under direct supervision when they met any difficulties in using the system. Therefore, more evaluation is needed before the system is ready for widespread use by a broad group of teachers. As for the learning

outcomes, a comparison group of students was not used. The reason is that the system is designed as an additional learning aid and not as a self-contained teaching method. In addition, the system was evaluated only in the domain of factoring and not the whole domain that the system covers. Finally, a feature of the system that has not been adequately evaluated is its fine-grained student model and the possible benefits of the detailed information it provides to both students and teachers.

In order to further investigate the reusability and expandability of the system, one could try to extend its domain model to teach algebraic operations of rational algebraic expressions. To simplify rational expressions, a student should make full use of the operations already taught by the MATHESIS Algebra Tutor. Implementing such a demanding task will be the best test for the knowledge reusability and implementation extensibility of the MATHESIS system.

Chapter 3

Chapter 3: The MATHESIS Meta-Authoring Framework

3.1 INTRODUCTION

The MATHESIS tutor forms a challenging landmark for existing authoring frameworks and their authoring tools, for the following reasons:

- a) To the best of the author's knowledge, there is no other model-tracing Algebra Tutor able to teach the expansion and factoring of *any* algebraic expression that contains *any* combination of the math skills (algebraic operations) covered by the MATHESIS tutor.
- b) None of the Algebra Tutors created so far features real time problem analysis, solution and tutoring.
- c) Supported by its intelligent task recognition feature, the MATHESIS tutor can be expanded with other algebraic operations like rational expressions, equations of first and second degree as well as rational equations.

- d) Current authoring frameworks and their authoring tools cannot support the authoring of such a tutor.

Despite the efforts, advancements and successes in the currently developed authoring frameworks and the corresponding tutors, these frameworks have worked around the knowledge acquisition problem rather than confronting it directly. As a consequence, most of the developed tutors suffer from limited depth and breadth, whereas those having broader and deeper domain expertise models suffer from scalability issues. This is the motivation to deal directly with the knowledge acquisition problem in order to produce tutors that cover broader and more complex domains in a scalable way.

The rest of the chapter is structured as follows: Section 3.2 presents the background of the thesis work consisting of an overview of the state-of-the-art in authoring frameworks, the tutors produced and how they suffer from the knowledge acquisition bottleneck, coupled with a description of how the MATHESIS meta-authoring framework provides the means to deal with this problem by using the results of research in the ontological engineering field. Section 3.3 presents an overview of the MATHESIS meta-authoring framework. Section 3.4 describes the key characteristic of the framework, *OntoMath*, a meta-knowledge engineering language for the representation of procedural authoring knowledge within the MATHESIS ontology as an executable authoring model. Finally, Section 3.5 presents the MATHESIS authoring and meta-authoring tools.

3.2 BACKGROUND

This Section presents an overview of the state-of-the-art in authoring frameworks and the tutors produced, focusing on the knowledge acquisition bottleneck issue. Then it shows how the MATHESIS meta-authoring framework provides the means to deal with this problem.

3.2.1 Related Work

The most successful and widely used math MTTs are Cognitive Tutors developed by Carnegie Learning¹¹, based on more than twenty years of cognitive science research at CMU (Koedinger & Corbett, 2006). Cognitive Tutors are now an integral part of complete curricula used in hundreds of middle and high schools throughout the United States. Cognitive Tutors have to adapt to very strict guidelines and educational goals of the US educational system, thus they are not designed to face the breadth, depth and scalability issues. Instead, they follow the textbook by teaching specific exercises that train the students in specific, simple cognitive skills that don't contain other sub-skills. Each problem has its own simple cognitive model and interface. Therefore, there is actually a set of independent tutors and not one tutor with a common cognitive model and interface. Concerning their scalability, the set of anticipated steps for a problem is *precomputed* by solving the problem in all acceptable ways by running a rule-based problem-solver (Van Lehn, 2006). Carnegie Learning uses a proprietary authoring tool, the Cognitive Tutor SDK (Blessing, Gilbert, Ourada & Ritter, 2009), which supports the development of cognitive models based on the ACT Theory of cognition (Anderson, 1993). Problem solving states are represented by a hierarchy of *goalnode* instances with

¹¹ www.carnegielearning.com

their *properties* and values, while problem solving steps are represented by a hierarchy of predicates that operate on the goalnodes. No information is given on how broad and deep these cognitive models can be or if they can be reused between the various tutors developed.

A publicly available set of authoring tools for Cognitive Tutors are the Cognitive Tutors Authoring Tools (CTAT¹²) developed at the Human-Computer Interaction Institute of Carnegie Mellon University (Aleven, McLaren, Sewall, & Koedinger, 2006). After 7 years of use, CTAT is the most mature and widely used authoring tool. It supports two types of tutors, cognitive tutors, which were described above, and *example-tracing tutors* (Aleven, McLaren, Sewall, & Koedinger, 2009). While cognitive tutors have a cognitive model, implemented as a set of production rules in Jess¹³, example-tracing tutors have a “generalized example” of the solution of a specific problem, implemented as a “behavior graph”, an acyclic graph where nodes represent problem-solving states and links represent problem-solving steps. Example-tracing tutors are authored using a programming-by-demonstration technique by creating initially a tutor interface for the targeted problem type through drag-and-drop techniques, then demonstrating through this interface the problem’s solution and finally editing, annotating and generalizing the resulting behavior graph. In the case of cognitive tutors, the last step demands the development of the cognitive model implemented as production rules in Jess by AI programmers.

ASTUS¹⁴ is a framework for domain independent model-tracing tutors’ development. It is designed to provide a knowledge representation language for the

¹² <http://ctat.pact.cs.cmu.edu>

¹³ <http://www.jessrules.com>

¹⁴ <http://astus.usherbrooke.ca>

development of the cognitive model richer than that of CTAT (Paquette, Lebeau, & Mayers, 2010). The purpose is to model domains from a pedagogical perspective rather than a cognitive one, allowing experimentation with varied pedagogical strategies. The framework is relatively new and the authoring language is not yet fully developed, with only a few tutors implemented and no authoring tools developed.

ASPIRE¹⁵ is an authoring framework for the development of constrained-based tutors (Mitrovic, Martin, Suraweera, Zakharov, Milik, & Hooland, 2009). These tutors do not use a cognitive model to trace the student's solution in a step-by-step basis, but they are equipped with a set of constraints that describe the forms of correct solution(s) for the tutored problem. In a comparative study between model-tracing and constraint-based tutors (Mitrovich, Koedinger, & Martin, 2003), the authors conclude that "*Model-tracing is an excellent choice for domains where appropriate problem solving strategies are well-defined, and where comprehensive feedback on them is desirable. On the other hand, CBM offers a workable alternative when such strategies are not available or appropriate, or there is too little time or resources to build a model-tracing knowledge base*". Therefore, in addition to the breadth and depth issue, constraint-based tutors cannot provide the granularity necessary for, e.g., an algebra tutor.

Whenever there is need for a broad and/or deep cognitive model, authors usually start from scratch and fall back to customized solutions. Two such examples are the Andes¹⁶ physics tutor (VanLehn, Lynch, Schulze, Shapiro, Shelby, Taylor, Treacy, Weinstein, & Wintersgill, 2005) and the Visual Classification Tutoring Framework (VCT) (Crowley & Medvedeva, 2006).

¹⁵ <http://aspire.cosc.canterbury.ac.nz>

¹⁶ <http://www.andestutor.org/>

Andes contains 356 physics problems (mechanics, electricity and magnetism) solved by a knowledge base of 550 physics rules. These rules comprise “major principles”, like Newton’s second law ($F = m \cdot a$), as well as “minor principles”, like mathematical and common sense justifiers. The creation and maintenance of such a broad, deep and granular cognitive model raises drastically the demands in expertise and time resources (VanLehn et al., 2005). As far as it concerns the development time, Andes itself took five years to be built, while its development was based on the Cascade (VanLehn, 1999) and Olae (VanLehn, Johnes, & Chi, 1992) projects. Finally, there were significant scalability problems, since in order to add a new rule to the cognitive model authors should re-inspect the whole model! (VanLehn et al., 2005)

The same findings hold for the Visual Classification Tutoring (VCT) framework, which generally supports the development of tutors for visual classification, but specialises in medical domains like radiology, haematology and pathology. The framework makes the best provision for accommodating broad, deep, granular and scalable cognitive models by using ontologies to represent separately generic models for the domain model, the task model and the pedagogic model. This generic framework was used to develop SlideTutor⁷, a model-tracing tutor for a sub-domain of inflammatory diseases of skin, covering 33 diseases with 50 different diagnostic features. Once again, the expertise and time costs are high: an expert pathologist in cooperation with a knowledge engineer must annotate each diagnostic case with the contained disease and its diagnostic features. Based on this information, the task model produces dynamic solution graphs that guide the student in his/her diagnosis.

The use of ontologies and semantic web services in the field of ITSs is relatively new. Ontological engineering is used to represent learning content, organize learning repositories, enable sharable learning objects and learner models and facilitate the reuse

of content and tools (Dicheva, Mizoguchi, & Greer, 2009). Examples of intelligent tutoring systems that use ontologies are *Activemath* (Melis, Andrès, Büdenbender, Frischauf, Gogvadze, Libbrecht et al. 2001), which uses ontological representation of mathematical concepts, learning goals and acquired knowledge, and *SlideTutor*¹⁷. However, these are intelligent tutoring systems and not authoring systems.

An ontology-based authoring system for constraint-based tutors is ASPIRE (Suraweera et al., 2009), which uses ontologies to define the concepts of the domain and then, based on these definitions, to provide the constraints for possible solutions used by the authored constraint-based tutors.

The most relevant work to the MATHESIS framework is the OMNIBUS/SMARTIES project (Mizoguchi, Hayasi, & Bourdeau, 2009). The OMNIBUS ontology is a heavy-weight ontology of learning, instructional and instructional design theories. Based on the OMNIBUS ontology, SMARTIES (SMART Instructional Engineering System) is a theory-aware system that provides a modelling environment and guidelines for authoring learning/instructional scenarios. While the OMNIBUS/SMARTIES system provides support mainly for the design phase of ITS building, the MATHESIS framework aims at the analysis and development phases. It provides a semantic description of both tutoring and authoring knowledge of any kind of tutor in the form of composite processes and the way to combine them as building blocks of intelligent tutoring systems. Thus, it provides the ground for achieving reusability, shareability and interoperability.

Although ASPIRE and OMNIBUS/SMARTIES are ontology-based authoring systems, they differ from MATHESIS framework being a meta-authoring system. These

¹⁷ <http://slidetutor.upmc.edu/>

systems provide specific authoring programs that use a *static* ontological representation of tutoring and authoring knowledge to build a *specific* kind of tutors. The MATHESIS framework provides meta-authoring tools and an authoring language for expert authors to write authoring programs in the form of executable OWL-S authoring processes. These authoring programs can then be *executed* by the authoring tools to guide less expert authors in generating the ontological representation of *any* kind of tutor. This ontological representation of the tutor can then be translated to program code.

3.2.2 Ontological Engineering and the Knowledge Gap Problem

The approach adopted in this thesis combines the research in the field of authoring tools for ITSs with the field of knowledge engineering tools for knowledge-based systems. This line of research starts with the first attempts to define reusable problem-solving knowledge through the introduction of the concepts of *Generic Tasks* (Chandrasekaran, 1986) and *heuristic classification* (Clancey, 1985). It continues with the concepts of *task ontologies* (Mizoguchi, Vanwelkenhuesen, & Ikeda, 1995) and the development of knowledge modeling frameworks like the MULTIS project (Mizoguchi, Vanwelkenhuesen, & Ikeda, 1995), the Protégé project (Puerta & Musen, 1992) and the KADS (Wielinga, Schreiber, & Breuker, 1992) and CommonKADS (Schreiber, et al., 1999) projects. The latter introduced the concept of Problem Solving Methods (PSMs). With the emergence of the Web, the necessity for representing and deploying PSMs in a shareable and reusable way led to their semantic (ontological) representation as Web Services. The ultimate goal is the development of knowledge-based systems from reusable knowledge components found on the web, a task known as *automated web*

service composition. Various frameworks with web services description languages have been developed, OWL-S being one of them. Although it is not an immediate intention of this thesis to view ITS authoring as a web service composition task, it sets the foundations, focusing on the shareability and reusability of authoring and tutoring knowledge provided by OWL-S.

Based on the success of the ontological engineering approach in the domain of expert systems (Aitken & Sklavakis, 1999; Lenat, 1995; Sklavakis, 1998), as well as in the domain of intelligent tutoring systems (Mizoguchi, Hayashi, & Bourdeau, 2009), two research goals were set:

- i. the complete ontological representation of a model-tracing tutor's modules, that is, the user interface, the tutoring model, the domain expertise model and the student model,
- ii. the complete ontological representation of the authoring knowledge that was used to build these models, and
- iii. the extensive use of standardized languages and publicly available modular tools.

For these reasons, a bottom-up approach was adopted: Initially, the MATHESIS Algebra Tutor was developed to be used as a prototype target tutor (Sklavakis & Refanidis, 2008). Then, based on the knowledge used to develop the Algebra Tutor, an initial version of the MATHESIS ontology has been developed using the Ontology Web Language - OWL¹⁸ (Sklavakis & Refanidis 2009b; Sklavakis & Refanidis, 2010b). As this first version of the ontology was developed in a bottom-up direction, it emphasized on the representation of the tutor's models, namely the interface, tutoring and domain

¹⁸<http://www.w3.org/TR/owl-features/>

expertise models. The ontology also contained a representation of the authoring knowledge at a rather conceptual level. At the final stage of the project, the generic meta-authoring tools were developed (Sklavakis & Refanidis, 2014). These tools include:

- i. An executable authoring language, `ONTOMATH`, based on the process model of `OWL-S`¹⁹,
- ii. editing tools for the development of `ONTOMATH` executable authoring expertise models, that is, an ontological representation of the declarative and procedural authoring knowledge, and
- iii. an interpreter for executing the `ONTOMATH` authoring models.

Using these tools, an authoring model was built that, when executed, builds the ontological representation of a model-tracing monomial multiplication tutor identical to the one contained in the original Algebra Tutor. In parallel, authoring tools for the development of model-tracing tutors have been developed. These tools are used to support the meta-authoring tools in the development of the executable authoring model by automating some top-level authoring processes of the MTT under development and providing visualisation and browsing facilities for the inspection of the tutor's developed models.

3.3 AN OVERVIEW OF THE MATHESIS META-AUTHORING FRAMEWORK

The MATHESIS framework is mainly a meta-knowledge engineering framework. It is well known that knowledge engineering is knowledge of how to extract problem-solving knowledge from domain experts, represent this knowledge in a suitable format and implement a system that uses this knowledge to solve problems like a human expert

¹⁹ <http://www.w3.org/Submission/OWL-S/>

(Aitken & Sklavakis, 1999; Lenat, 1995; Sklavakis, 1998). In the case of authoring systems for ITSs, a meta-authoring framework should enable knowledge engineers (meta-authors) to extract authoring knowledge from expert authors, that is, cognitive scientists and programmers (Artificial Intelligence or general purpose); represent authoring knowledge in a suitable format; and implement a system that uses this knowledge to guide authors of lower levels of expertise to build tutoring systems. To achieve these three objectives, the MATHESIS meta-authoring framework adds a *semantic level* on top of the knowledge level of each authoring framework (Figure 3.1). Its purpose is to represent declaratively (ontologically) the authoring expertise used to build ITSs, now lying partially unexpressed into the heads of authoring experts and partially expressed into the authoring tools, as well as the authored tutoring knowledge hard-wired into the ITSs themselves.

The key point of the proposed framework is the *ontological declarative representation* of these two kinds of knowledge. At the same time, and that was the most challenging problem, these declarative representations should also be executable. More specifically, the deployment of the framework is done in the following stages (Figure 3.1, bottom to top):

1. A knowledge engineer specialized in the MATHESIS framework (meta-author), extracts the authoring expertise from the domain experts, that is, cognitive scientists and AI programmers. The expertise must cover all stages of ITS development, that is, analysis, design and implementation. This constitutes a crucial difference between the framework-specific authoring tools described in the previous Section and the objectives of the MATHESIS framework: the former support parts of the ITS development stages, usually leaving out the most difficult

- ones like the analysis stage, while the latter allows meta-authors to encode authoring knowledge of any stage.
2. Using the meta-authoring tools, the meta-author creates an executable ontological model of the extracted authoring knowledge, the *authoring expertise model*. This model contains authoring processes of ONTOMATH, a special purpose language developed within the framework. ONTOMATH defines two kinds of authoring processes: (a) *composite authoring processes*, which correspond to the functions/procedures of a programming language and are represented using the process model of OWL-S, and (b) *atomic authoring processes*, which correspond to the statements of a programming language. When the authoring model is executed by a non-expert author (e.g. domain expert), the ONTOMATH interpreter executes them by calling corresponding Java methods that in turn use the Protégé API to guide the non-expert author in building the ontological representation of the ITS models (cognitive, tutoring, interface) into the MATHESIS ontology. Therefore, the authoring processes are the semantic representation of the framework-specific authoring tools.
 3. The ontological representation of the ITS's various models (cognitive, teaching, interface) contain both declarative and procedural knowledge. An example of declarative knowledge would be the interface structure (interface model) or the problem-solving concepts and stages of the cognitive model. An example of procedural knowledge would be the model-tracing algorithm (tutoring model) or the problem-solving steps of the cognitive model. In the MATHESIS framework these knowledge elements are defined by the meta-author as generic elements. Declarative knowledge elements are defined using the common OWL structures: classes, instances, properties and values. Procedural knowledge elements are

- defined using the process model of OWL-S, just like the composite authoring processes described in stage 2. It is these generic knowledge elements that the executed authoring processes act on, guiding the non-expert author to create specific-ones for the tutor under development.
4. The meta-author may develop framework-specific (e.g. model tracing) authoring tools to help himself develop the authoring model and the non-expert authors in developing the tutor(s). These are mainly visualisation tools, although they can also provide manual creation and editing of tutor-specific knowledge elements based on generic ones. This last facility aims at accommodating more expert authors that can develop parts of the tutor directly, without executing the corresponding authoring processes. A suite of such framework-specific tools for model-tracing tutors has been developed.
 5. Having created the ontological representation of the tutor, the non-expert author can create its implementation by translating the ontological model to specific programming languages. For example, in the case of the MATHESIS Algebra Tutor, the interface model is translated to HTML and the cognitive and tutoring models to JavaScript. These translations are performed automatically by special translation tools. In case of other target programming languages, we need to develop its corresponding ontological representation as well as the translation tool.

All stages are performed using the MATHESIS tools (Figure 3.2)

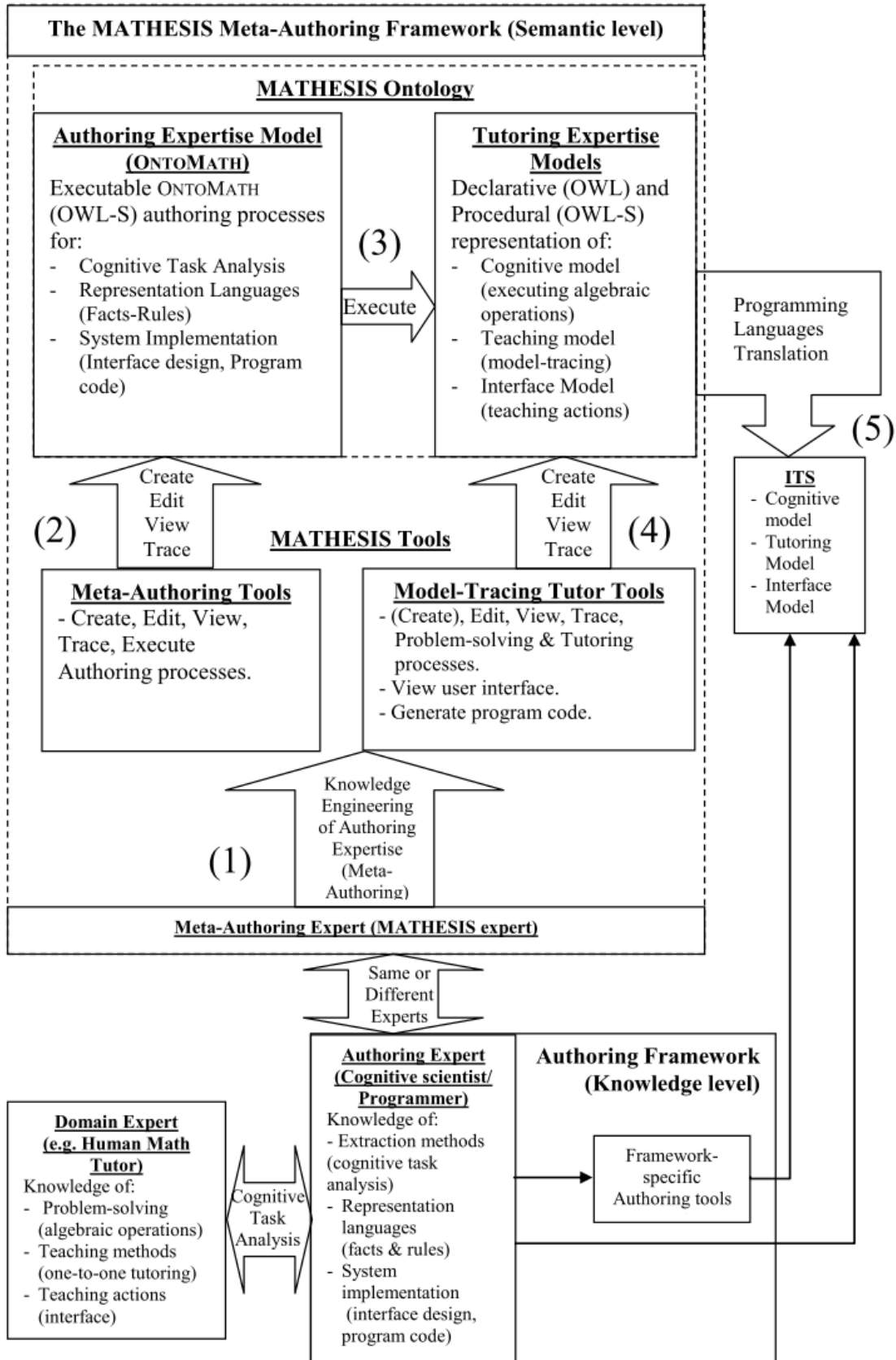


Fig. 3.1 The MATHESIS Meta-Authoring Framework

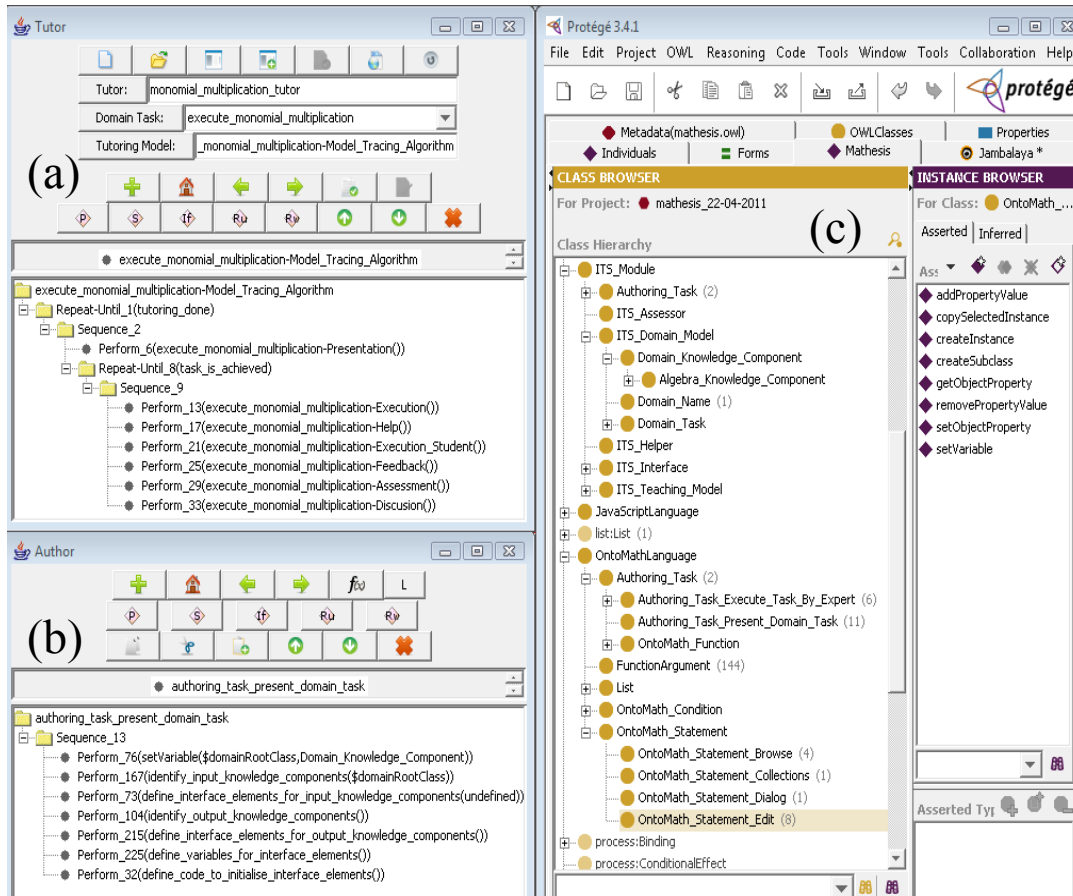


Fig. 3.2. The MATHESIS Tools as a tab widget in Protégé: (a) Framework-specific (model-tracing) Tutor Authoring Tools, (b) Authoring Processes (Meta-Authoring) Tools, (c) The MATHESIS Ontology Tab

3.4 THE ONTOMATH META-KNOWLEDGE ENGINEERING LANGUAGE

The main component of the MATHESIS authoring framework is the MATHESIS Ontology. It contains three kinds of knowledge:

- i. The declarative knowledge of the tutor, such as the interface structure and the problem-solving concepts and stages of the cognitive model,

- ii. the procedural knowledge of the tutor, such as the teaching and math domain expertise models and, finally,
- iii. the authoring knowledge, that is, the declarative and procedural knowledge that is needed to develop the tutor.

3.4.1 Procedural Knowledge Representation: The OWL-S Process Model

While the declarative knowledge is represented with the basic OWL components, the procedural knowledge, both tutoring and authoring, is represented via the *process model* of the OWL-S web services description ontology. Through OWL-S, every authoring or tutoring task is represented as an authoring or tutoring process, composite or atomic.

Using the OWL-S process model to represent ontologically procedural knowledge, like teaching, math problem-solving or authoring knowledge, is the key advantage of the MATHESIS framework that gives a new perspective in the development of reusable authoring knowledge for intelligent tutors. OWL-S is a web service description ontology designed to enable the following tasks:

- Automated discovery of Web services that can provide a particular class of service capabilities, while adhering to some client-specified constraints.
- Automated Web service invocation by a computer program or agent, given only a declarative description of the service.
- Automated Web service selection, composition and interoperation to perform some complex task, given a high-level description of an objective.

The last task is of interest for the MATHESIS framework and therefore the focus will be set on it. To support this task, OWL-S provides, among other things, a language for describing service compositions as seen in Figure 3.3 (Martin et al., 2005). Every service is viewed as a *process*. OWL-S defines *Process* as a subclass of *ServiceModel*. There are three subclasses of *Process*, namely the *AtomicProcess*, *CompositeProcess* and *SimpleProcess*. Atomic processes correspond to the actions a service can perform by engaging it in a single interaction. In the MATHESIS ontology they represent simple statements, either tutoring or authoring, grounded to JavaScript or Java code correspondingly. Composite processes correspond to actions that require multi-step protocols. In the MATHESIS ontology they represent functions, either tutoring or authoring, that call other functions (composite processes). Finally, simple processes provide an abstraction mechanism to provide multiple views of the same process. Currently, they are not used in the MATHESIS framework.

Composite processes are decomposable into other composite or atomic processes. Their decomposition is achieved by using control constructs such as *Sequence* or *If-Then-Else*. Table 3.1 shows the most common control constructs that OWL-S supports.

Any composite process can be considered as a tree whose non-terminal nodes are labelled with control constructs. The leaves of the tree are invocations of other processes, composite or atomic. These invocations are indicated as instances of the *Perform* control construct. This special control construct takes as a parameter a process, either composite or atomic. In the MATHESIS framework a *Perform* with an atomic process corresponds to the execution of a statement, whereas a *Perform* with a composite process corresponds to calling a function. This tree-like representation of composite processes is the key characteristic of the OWL-S process model and has been used in the MATHESIS ontology to represent both authoring and tutoring procedural knowledge.

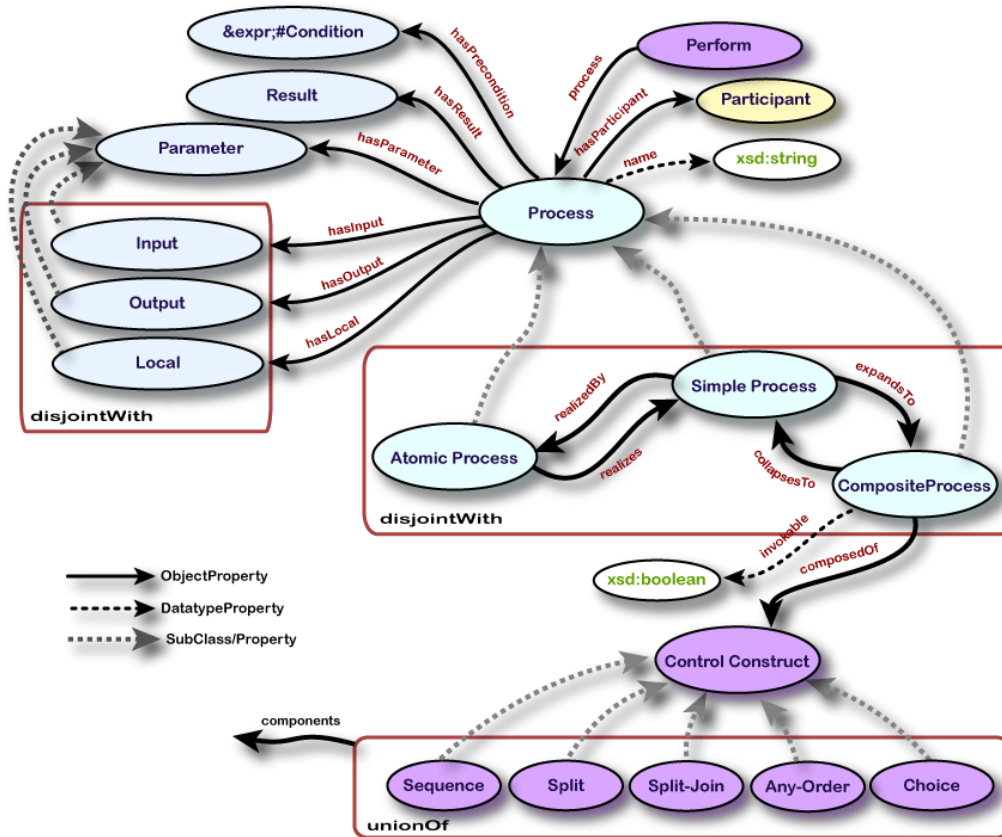


Fig. 3.3. Top level of the OWL-S process ontology (from Martin et al., 2005)

Table 3.1. Common control constructs supported by the OWL-S process model

Control Construct	Description
Sequence	A list of control constructs to be performed in order
Choice	Calls for the execution of a single construct from a given bag of control constructs (given by the components property). Any of the given constructs may be chosen for execution
If-Then-Else	It has properties <i>ifCondition</i> , <i>then</i> and <i>else</i> holding different aspects of the If-Then-Else construct
Repeat-While & Repeat-Until	The initiation, termination or maintenance condition is specified with a <i>whileCondition</i> or an <i>untilCondition</i> respectively. The operation of the constructs follows the familiar programming language conventions.

3.4.2 Procedural Authoring Knowledge Representation: The ONTOMATH language

The MATHESIS framework allows expert meta-authors to capture the whole authoring effort by providing an *executable authoring model building* language, namely ONTOMATH. In ONTOMATH, each authoring step is represented as an authoring process, composite or atomic (Figure 3.4). Composite authoring processes correspond to functions of a programming language that can be called, get and return values. This is achieved by two means: a) Each composite process has a property, `hasFormalParameters`, which keeps a list of the process formal parameters, and b) each `Perform` has a property, `hasRealParameters`, which keeps the list of the parameters at call time. During execution of a `Perform` construct by the authoring tools, the interpreter matches the values of the real parameters to those of formal parameters. The values of the two properties, `hasFormalParameters` and `hasRealParameters`, are defined by the meta-author in the ontology with the help of the meta-authoring tools. The recursive analysis of composite authoring processes ends to atomic authoring processes, which are instances of the `OntoMathStatement`, a subclass of `AtomicProcess`. Each `OntoMathStatement` instance corresponds to an operation that must be performed to the MATHESIS Ontology. Table 3.2 presents a concise description of the ONTOMATH authoring statements. For a complete description, see Appendix B.

ONTOMATH authoring processes, composite and atomic, are classified in classes and subclasses according to the kind of authoring knowledge they represent and the part of the authoring endeavour they implement. For example, in Figure 3.4, there are ONTOMATH processes (`identify-input-knowledge-components`, `define-interface-elements-for-input-knowledge-components`, `define-variables-for-interface-elements`, `define-code-to-initialise-interface-elements`) for implementing the authoring task of the problem presentation tutoring task (class `Authoring-Task-Present-Domain-Task`). Other ONTOMATH processes, like `get-HTML-`

Element-Property, serve the authoring task of HTML programming (class Programming-Task-HTML). Others, like get-interface-element-reference, serve the authoring task of JavaScript programming (class Programming-Task-JavaScript). As a consequence, the ONTOMATH language is not only capturing authoring knowledge but, at the same time, is *classifying* authoring knowledge thus enabling reasoning on that knowledge, that is, discovery, retrieval, reuse and modification, by authors of different expertise levels.



authoring_task_present_domain_task

define_code_to_initialise_interface_elements
 define_global_user_interface_container
 define_interface_elements_for_input_knowledge_components
 define_interface_elements_for_output_knowledge_components
 define_variables_for_interface_elements
 identify_input_knowledge_components
 identify_output_knowledge_components

getHTMLElementProperty

get_interface_element_reference

getSelectedClass
 getSelectedInstance
 setSelectedClass
 setSelectedInstance

copySelectedInstance
 createInstance
 createSubclass
 getDataProperty
 getInstance
 getLocalName
 getObjectProperty
 setDataProperty
 setObjectProperty
 setVariable

Fig. 3.4. Part of the ONTOMATH Authoring Processes Ontology

Table 3.2. The ONTOMATH Statements and their operations

<u>Browse Statements</u>	Purpose
<i>setSelectedClass(className)</i>	Sets the Class named by <i>className</i> as selected in the Classes Panel
<i>getSelectedClass(className)</i>	Sets <i>className</i> to the name of the Class selected in the Classes Panel
<i>setSelectedInstance(instanceName)</i>	Sets the Instance named by <i>instanceName</i> as selected in the Instances Panel
<i>getSelectedInstance(instanceName)</i>	Sets <i>instanceName</i> to the name of the Instance selected in the Instances Panel
<u>Collection Statements</u>	Purpose
<i>iteratorNext(iterator, element)</i>	Sets <i>element</i> to the next element of <i>iterator</i>
<u>String Statements</u>	Purpose
<i>strConcat(newString, string1, string2)</i>	Concatenates <i>string1</i> and <i>string2</i> and returns the concatenated string to <i>newString</i>
<u>Dialog Statements</u>	Purpose
<i>showMessageDialog(message)</i>	Displays a Message Dialog with <i>message</i>
<i>showInputDialog(userInput, message, defaultValue)</i>	Displays an Input Dialog with <i>message</i> and <i>defaultValue</i> . Returns user input to <i>userInput</i>
<u>Ontology Editing Statements</u>	Purpose
<i>createInstance(instanceName, className)</i>	Creates a new instance of class <i>className</i> named <i>instanceName</i>
<i>copySelectedInstance(newInstanceName)</i>	Creates a new copy of the instance that is currently selected in the Instances Panel with name <i>newInstanceName</i> .
<i>createSubclass(subclassName, superclassName)</i>	Creates a new subclass named <i>subclassName</i> of class <i>superclassName</i>
<i>getObjectProperty(instance, property, propertyValues)</i>	Gets the values of object property <i>property</i> of instance <i>instance</i> and stores them to variable <i>propertyValues</i> .
<i>setObjectProperty(instance, property, value)</i>	If object property of <i>instance</i> is functional (takes only one value), its value is set to <i>value</i> . Otherwise, <i>value</i> is added to the list of the property's values.
<i>getDataProperty(instance, property, propertyValues)</i>	Gets the values of data property <i>property</i> of instance <i>instance</i> and stores them to variable <i>propertyValues</i> .
<i>setDataProperty(instance, property, value)</i>	If data property of <i>instance</i> is functional (takes only one value), its value is set to <i>value</i> . Otherwise, <i>value</i> is added to the list of the property's values.
<i>removePropertyValue(instance, property, value)</i>	Removes instance <i>value</i> from the value(s) of property

	<i>property</i> of instance <i>instance</i> .
<i>getLocalName(instance, instanceName)</i>	Gets the local name of instance <i>instance</i> and stores it in variable <i>instanceName</i> .
<i>setVariable(variable, value)</i>	Sets the value of variable <i>variable</i> to <i>value</i> .
<u>Tutoring Processes Editing Statements</u>	Purpose
<i>addPerform(performStatement)</i>	Adds a Perform Control Construct (tutoring statement) as a child to the currently selected construct in the displayed tutoring process tree

ONTOMATH statements are *grounded* to actual Java program code. When the MATHESIS authoring tools interpret a Perform construct that calls an ONTOMATH statement, they execute its corresponding Java code, which performs the corresponding operations on the ontology that represents the tutor under development. It must be noted that the set of ONTOMATH statements is not fixed. Expert meta-authors can define their own atomic authoring (ONTOMATH) statements by using the meta-authoring tools to define in the ontology the values of property `hasFormalParameters` for the new statement and writing the Java code that, during execution, gets the values of property `hasRealParameters` of the calling Perform construct and performs the statement's intended operation(s). The interpretation and execution of the ONTOMATH code by the MATHESIS authoring tools leads to the creation of the tutor's ontological representation and, consequently, to the implementation of the authored tutor

Therefore, the ONTOMATH authoring processes form an ontological representation of a *meta-program* that handles the ontological representation of the tutor as its data. They capture the authoring expertise of expert meta-authors and make it available to non-expert authors. They constitute the expertise model of an ITS meta-authoring shell that, when executed, guides non-expert authors to develop their own tutors.

3.5 THE MATHESIS AUTHORING TOOLS

This section describes the use of the MATHESIS authoring tools. The tools have been implemented as a tab widget in Protégé (Figure 3.2). The tools are grouped in three windows according to their functionality: (a) Framework-specific (model-tracing) Tutor Authoring Tools, (b) Authoring Processes (Meta-Authoring) Tools, and (c) The MATHESIS Ontology Tab. This section presents an overview of the tools. An extensive description of their use is presented in the next section. For this reason, this section contains some references to figures of the next section.

3.5.1 The Tutor Authoring Tools

The Tutor authoring tools (Figure 4.1) are used both by less-expert domain authors as well as by expert meta-authors. They are organized in three parts: (a) The Tutor Initialization Tools, (b) The Advanced Tools for Tutoring Processes Authoring, and (c) The Tutoring Processes Display/Editing area.

The Tutor Initialization Tools are shown in Figure 3.5. Tools 1-5 (*Create Tutor, Open Tutor, Create Domain Task, Add Domain Task and Select Tutoring Model*) will be described in detail in the next section. The rest of the tools are:

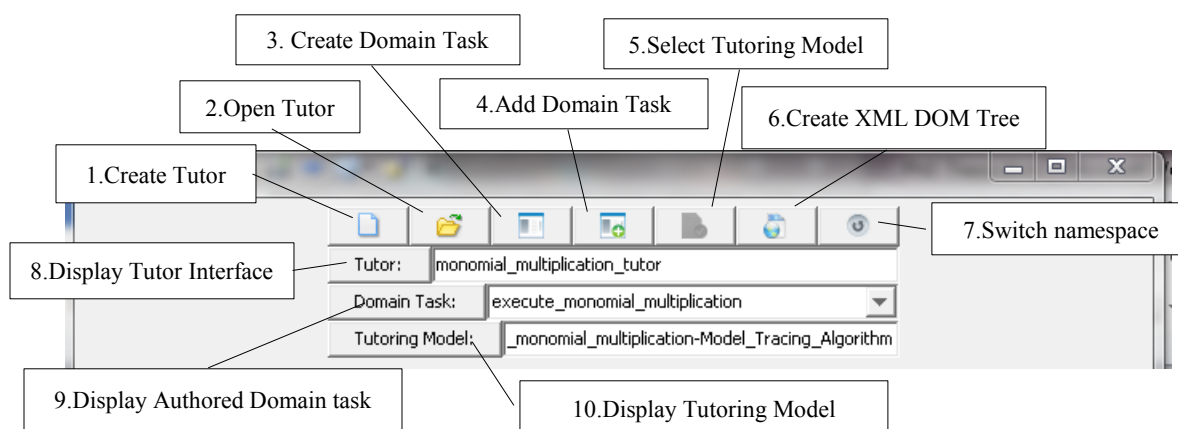


Fig. 3.5 The Tutor Initialization Tools

6-Create XML DOM Tree: Creates an XML Document Object Model tree from an XML or HTML file. An Open File dialogue asks from the author to locate a file containing a web page. Then the file is parsed and an XML DOM tree is created. From this XML tree, the ontological representation of the tutor's interface DOM can be created (not implemented). Figure 3.6 shows (part of) the XML DOM tree generated from the MATHESIS Algebra Tutor web page file.

7-Switch Namespace: This tool toggles the ontology namespace between this of the currently developed tutor, e.g. <http://users.uom.gr/~dsklavakis/monomial-multiplication-tutor#monomial-multiplication-tutor>, and that of the MATHESIS ontology. This switching is necessary when a meta-author needs to add a generic element in the MATHESIS ontology while developing a specific tutor.

8-Display Tutor Interface: Displays the tutor's interface DOM tree (Figure 4.11, right top) in the Tutoring Process Display/Edit Area (Figure 4.1c). It also sets the focus on the tutor's instance in the MATHESIS ontology.

9-Display Authored Domain Task: Sets the focus on the instance of the currently authored domain task in the MATHESIS ontology.

10-Display Tutoring Model: Displays the Tutoring Model process in the Tutoring Process Display/Edit Area (Figure 4.1c).

The Advanced Tools for Tutoring Processes Authoring are shown in Figure 3.7. These tools are used by experienced meta-authors to directly create hard to build tutoring processes – without using authoring processes –and provide them as libraries to domain authors. The most characteristic example of such a tutoring process is the Tutoring Model of the tutor, the `ModelTracingAlgorithm` tutoring process, described in the previous section. The use of the tools will be exemplified in Section 4 by describing how they are used by a meta-author to create/edit the generic `ModelTracingAlgorithm` tutoring process.



Fig. 3.6 The XML DOM tree of the MATHESIS Algebra Tutor interface

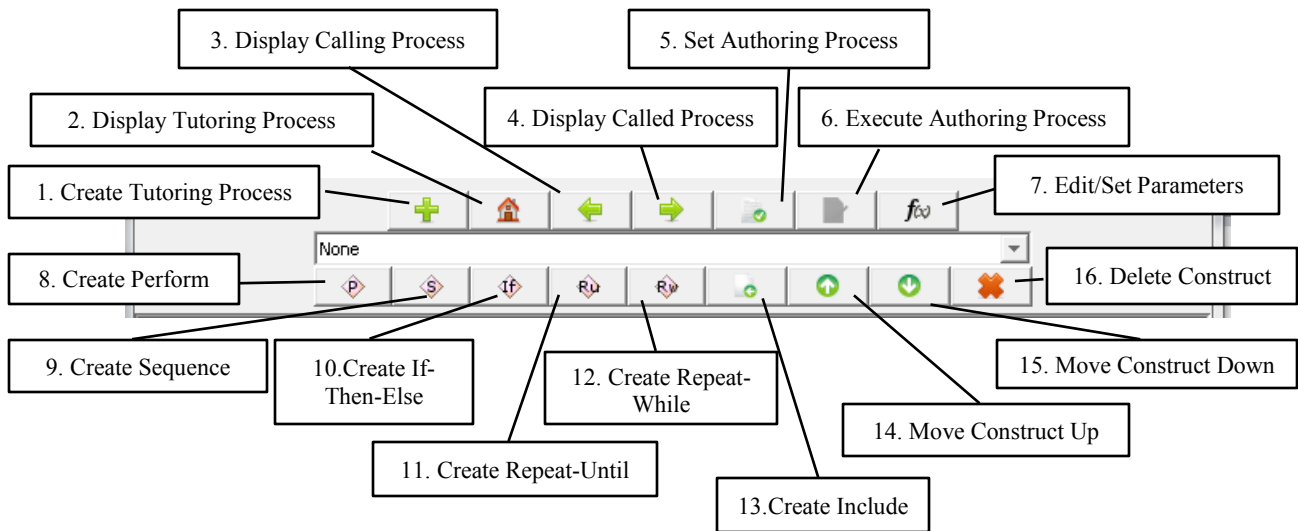


Fig. 3.7 The Tutoring Processes Advanced Authoring Tools

1-Create Tutoring Process: The (meta-) author selects a class in the CLASS BROWSER of the MATHESIS ontology window. In the case of the ModelTracingAlgorithm tutoring process, this is class ITS-Teaching-Model (Figure 4.7a). By clicking the button the system asks for the number of the process' parameters and a new instance of the class is created and displayed in the Display/Edit area (Figure 3.8).

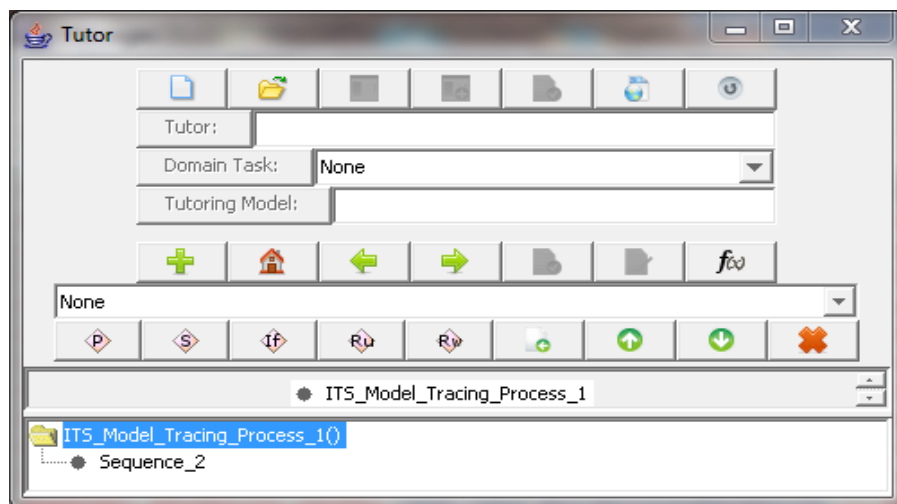


Fig. 3.8 A newly created Tutoring Process

2-Display Tutoring Process: Displays the Tutoring Process selected by the author in the MATHESIS ontology window (Figure 4.7). It also displays the tutoring process as the root of the call sequence tree just above the Display/Edit area (Figure 4.1c).

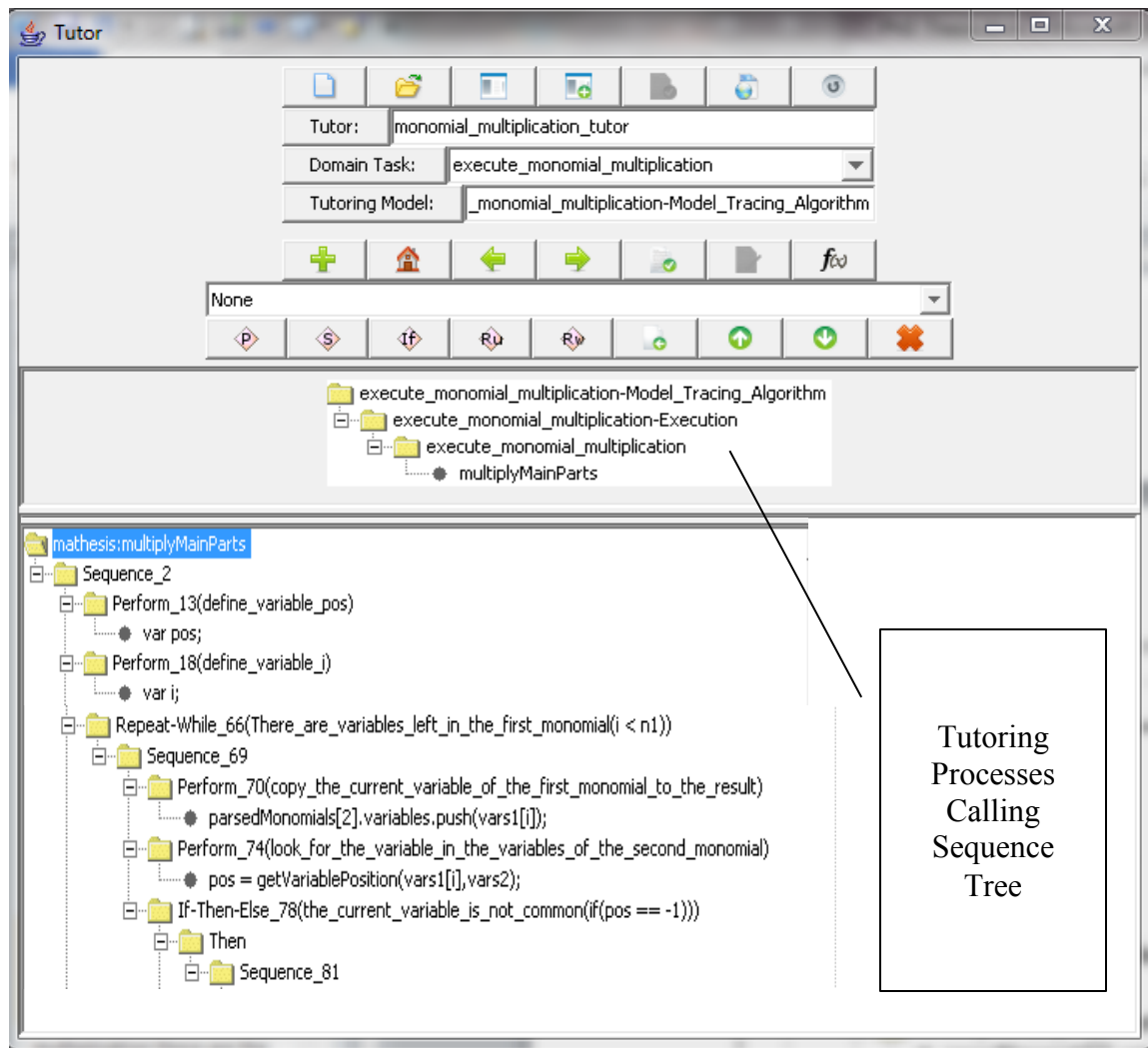


Fig. 3.9 The Calling Sequence Tree for Tutoring Process multiplyMainParts

3-Display Calling Process: Displays (pops) in the Display/Edit area the tutoring process that calls (parent) the currently displayed tutoring process (child). Figure 3.9 shows the calling sequence tree for tutoring process multiplyMainParts. The process is called

by process `execute-monomial-multiplication`. Clicking the *Display Calling Process* button would display the calling (parent) tutoring process, `execute-monomial-multiplication` and remove the `multiplyMainParts` process from the calling sequence tree.

4-Display Called Process: When the author selects a Perform that calls (parent) a composite tutoring process (not a JavaScript statement) the called process (child) is displayed in the Display/Edit area. The called process is added as a child of the calling process in the calling sequence tree. This tool, in combination with the *Display Calling Process* tool, allows authors to browse the tutor's code (tutoring processes).

5-Set Authoring Process: As explained before, each tutoring process can have a corresponding authoring process that, when executed, guides authors in creating the tutoring process. The meta-author selects the authoring process in the MATHESIS ontology BROWSER and clicks the button to associate it with the tutoring process displayed.

6-Execute Authoring Process: When a tutoring process has an associated authoring process this button displays the authoring process in the Display/Edit/Execute area of the Meta-Authoring Tools. In Figure 4.1 tutoring process `execute-monomial-multiplication-Presentation` (Perform-6) is associated with authoring process `authoring-task-present-domain-task`. Clicking the *Execute Authoring Process* button displays the `authoring-task-present-domain-task` process in the Display/Edit/Execute area of the Meta-Authoring Tools (Figure 4.12).

7-Edit/Set Parameters: This tool is used in two ways. If the name (root node) of a tutoring process is selected in the Display/Edit area is selected, the tool asks form the (meta-) author to set the formal parameters of the process. The author must enter the number of the parameters and for each one, an identifier and a short description (optional). If a Perform is selected, the tool asks form the (meta-) author to enter the

identifiers and descriptions of the real parameters. As a default, the tool proposes the identifiers and descriptions of the called process' formal parameters.

The remaining tools (8-16) are for editing the OWL-S constructs (program structures) that form the tutoring processes' code:

8-Create Perform to *12-Create Repeat-While*: They create the corresponding OWL-S constructs. The author selects an already existing construct and then clicks the corresponding button to create a new construct as the next sibling (tree node) of the selected construct.

13-Create Include is a special purpose (not OWL) construct that creates *include* statements for target programming languages that use this directive to load libraries, like JavaScript, C, Java etc.

14-Move Construct Up: Moves upwards the selected construct by switching it with its previous sibling construct (tree node).

15-Move Construct Down: Moves downwards the selected construct by switching it with its next sibling construct (tree node).

16- Delete Construct: Deletes the selected construct.

3.5.2 The Authoring Processes (Meta-Authoring) Tools

The Authoring processes authoring tools (Figure 4.12) are used by expert meta-authors to develop the authoring processes as well as by domain authors to execute them. Since authoring processes are OWL-S services, they are composed by the same OWL-S structures (Perform, Sequence, If-Then-Else, Repeat-Until, Repeat-While). Consequently, the authoring tools are almost identical with those for the tutoring processes. As seen in Figure 4.12, the layout of the tools is similar to that of the tutoring processes authoring

tool. At the upper part lie the button tools for displaying, editing and executing authoring processes (Figure 3.10). In the middle lies the calling stack represented as a tree and at the bottom lies the Display/Edit/Execute area.

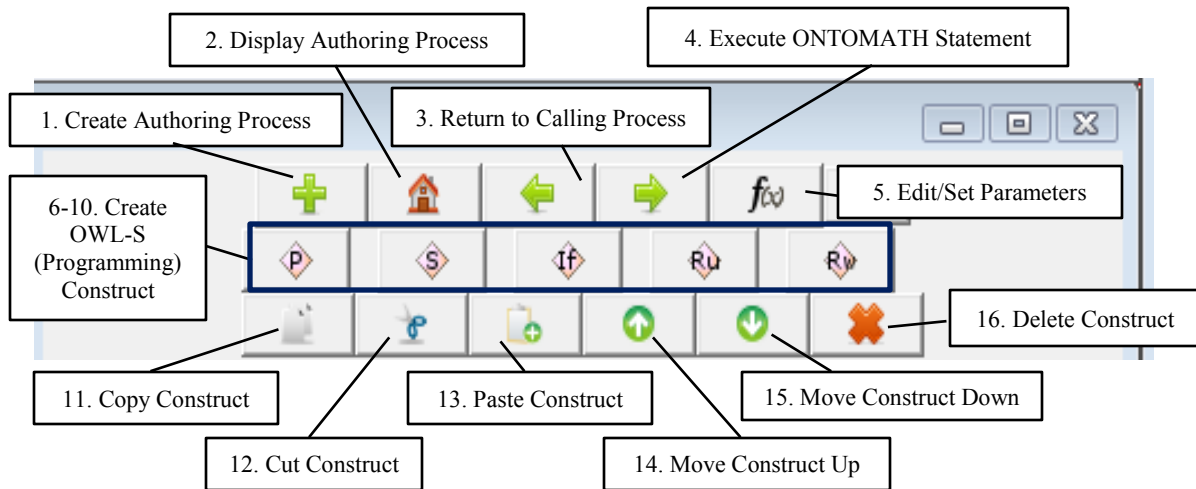


Fig. 3.10 The Authoring Processes (Meta-Authoring) Tools

1-Create Authoring Process: The meta-author selects a class in the CLASS BROWSER of the MATHESIS ontology window. By clicking the button the system asks for the number of the process' parameters and a new instance of the class is created and displayed in the Display/Edit area.

2-Display Authoring Process: Displays the Authoring Process selected by the meta-author in the MATHESIS ontology window. It also displays the authoring process as the root of the call sequence tree just above the Display/Edit/Execute area.

3-Return to Calling Process: This tool is part of the ONTOMATH language interpreter. It returns execution control to the calling authoring process. It displays (pops) in the Display/Edit/Execute area the authoring process that has called (parent) the currently executed authoring process (child).

4-Execute ONTOMATH Statement: This tool executes the currently selected ONTOMATH statement. If it is a Perform with a composite authoring process, it pushes the process onto the calling stack, displays the called process in the Display/Edit/Execute area and continues execution with it. If it is a Perform with an atomic authoring process, the process is interpreted and the corresponding Java code is executed to perform its operation. In case of another construct (Sequence, If-Then-Else, Repeat-Until, Repeat-While), the appropriate interpretation of the structure is achieved through the interpreter's Java code.

5-Edit/Set Parameters: This tool is used in two ways. If the name (root node) of an authoring process is selected in the Display/Edit/Execute area is selected, the tool asks from the meta-author to set the formal parameters of the process. The author must enter the number of the parameters and for each one, an identifier and a short description (optional). If a Perform is selected, the tool asks from the meta-author to enter the identifiers and descriptions of the real parameters. As a default, the tool proposes the identifiers and descriptions of the called process' formal parameters.

6-10-Create OWL-S Programming Construct: These tools create the corresponding OWL-S constructs. The author selects an already existing construct and then clicks the corresponding button to create a new construct as the next sibling (tree node) of the selected construct.

11-Copy Construct: Copies the selected construct

12-Cut Construct: Cuts the selected construct

13-Paste Construct: This tool is used in combination with the previous two tools. It pastes to the selected construct the construct previously copied or cut. If the selected construct is a Perform then the pasted construct is added as the next sibling. If the selected construct is a Sequence, If-Then-Else, Repeat-Until or Repeat-While, then the pasted construct is added as a child to the selected construct.

14-Move Construct Up: Moves upwards the selected construct by switching it with its previous sibling construct (tree node).

15-Move Construct Down: Moves downwards the selected construct by switching it with its next sibling construct (tree node).

16- Delete Construct: Deletes the selected construct.

Chapter 4

Chapter 4: Tutor Authoring in the MATHESIS Framework

4.1 INTRODUCTION

This Section describes how meta-authors create an authoring knowledge model using as an example the meta-authoring of a model-tracing tutor for monomial multiplication, as well as how non-expert domain authors can create the monomial multiplication tutor by executing the authoring model. The meta-authoring model for the monomial multiplication tutor was based on the authoring knowledge that was used to create the monomial multiplication part of the MATHESIS Algebra Tutor. Provision has been taken so that both the authoring model and the tutor's model can be extended for meta-authoring all the math skills tutored by the MATHESIS Algebra Tutor.

In the MATHESIS framework authors perform all authoring steps by executing the corresponding authoring processes created by meta-authors. However, special Tutor Initialization Tools have been developed, lying at the top of the Model-Tracing Tutor Authoring Tools window (Figure 4.1a). Instead of executing authoring processes, authors

use these tools to perform common authoring steps for the initialization of the tutor as it is described below.

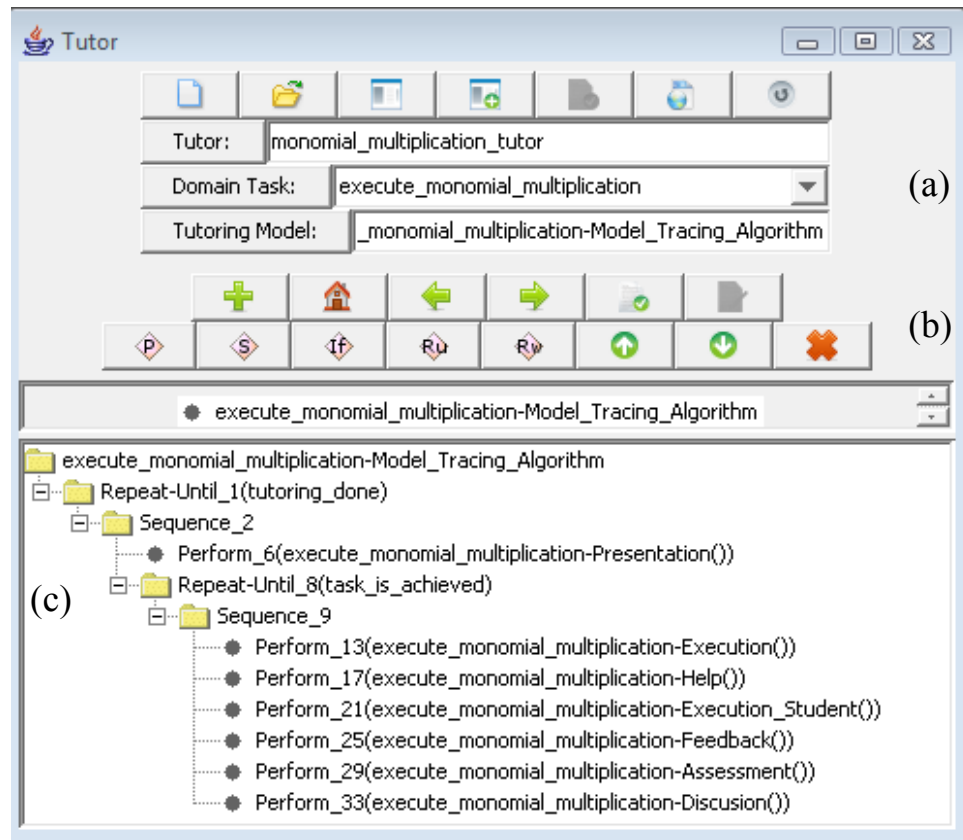




Fig. 4.1. The Model-Tracing Tutor Authoring Tools Window: (a) The Tutor Initialization Tools, (b) The Advanced Tools for Tutoring Processes Authoring, (c) Tree representation of tutoring process Model-Tracing-Algorithm for the execute-monomial-multiplication task

4.2 TUTOR INITIALIZATION²⁰

The first authoring action is to define an instance of the tutor in the ontology. At the top level of the MATHESIS ontology, every tutor is represented as an instance of class ITS-Implemented. In Figure 4.2 this instance is `monomial_multiplication_tutor`. The author

²⁰ http://ai.uom.gr/dsklavakis/en/mathesis/kes2011/01-Authoring_Tools.mp4

creates this instance once in the first authoring session using the “Create New Tutor” button  (Figure 4.1a). The author is prompted to enter the tutor’s name (Figure 4.3). In subsequent authoring sessions the author selects the tutor to edit using the “Open Existing Tutor” button . In both cases, the Tutor Initialization Authoring Tools (Figure 4.1a) automatically select the ITS_Implemented class in the ontology tab (Figure 4.4).

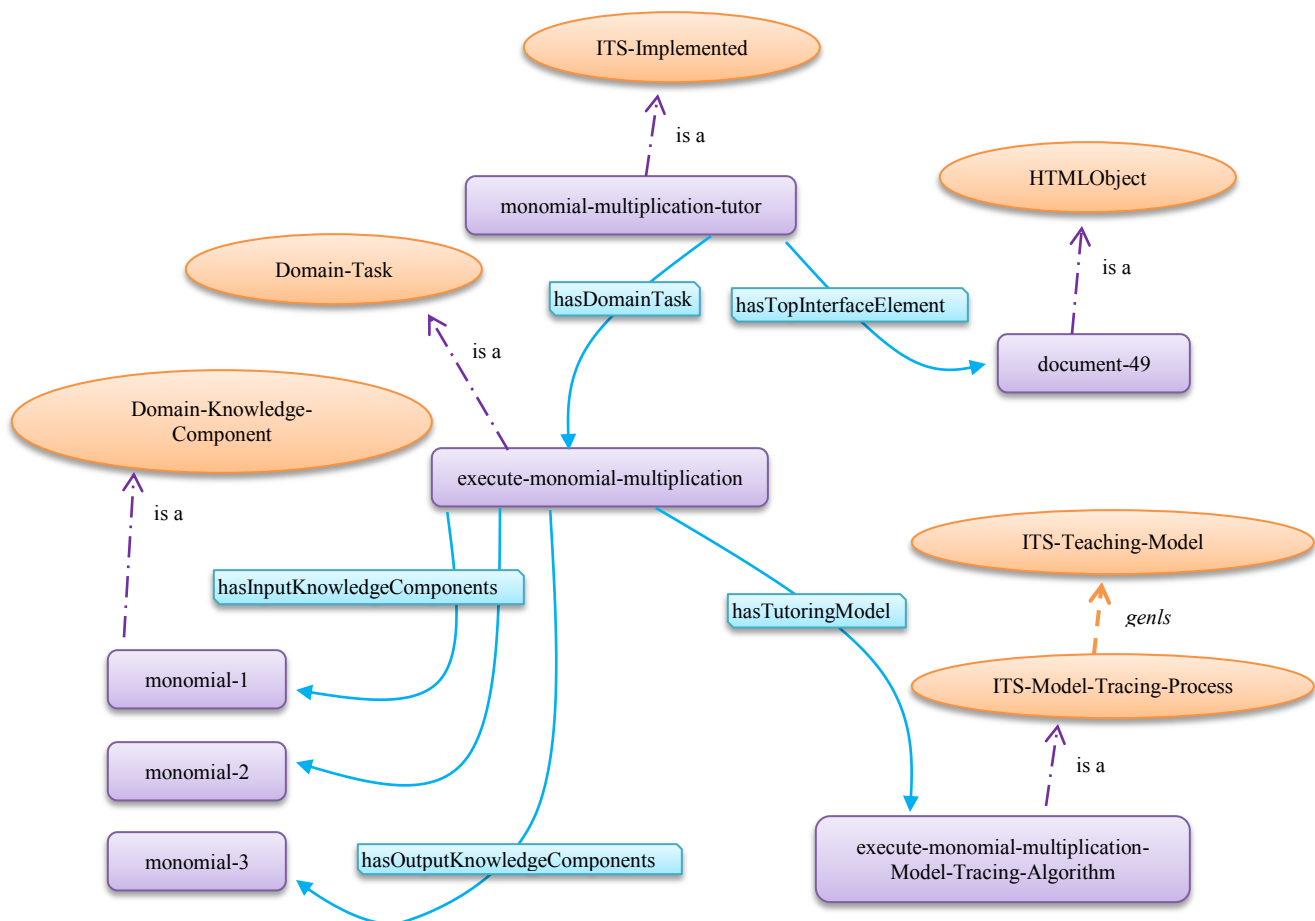


Fig. 4.2. The top-level ontological representation of the tutor

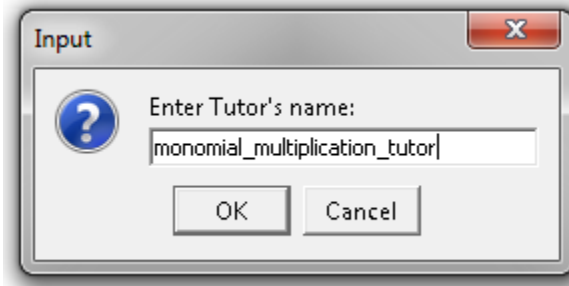


Fig. 4.3. Author is prompted by the tools to enter the name of a newly created tutor instance.

Class ITS-Implemented is a top class created by the meta-author and can have subclasses like the ITS-Implemented_Algebra subclass (Figure 4.4a), which are used for classifying the various tutors. This classification can use various criteria defined by the meta-author(s) such as the tutors' domain (math, physics, programming), thus making easier for authors to locate a specific tutor in the ontology.

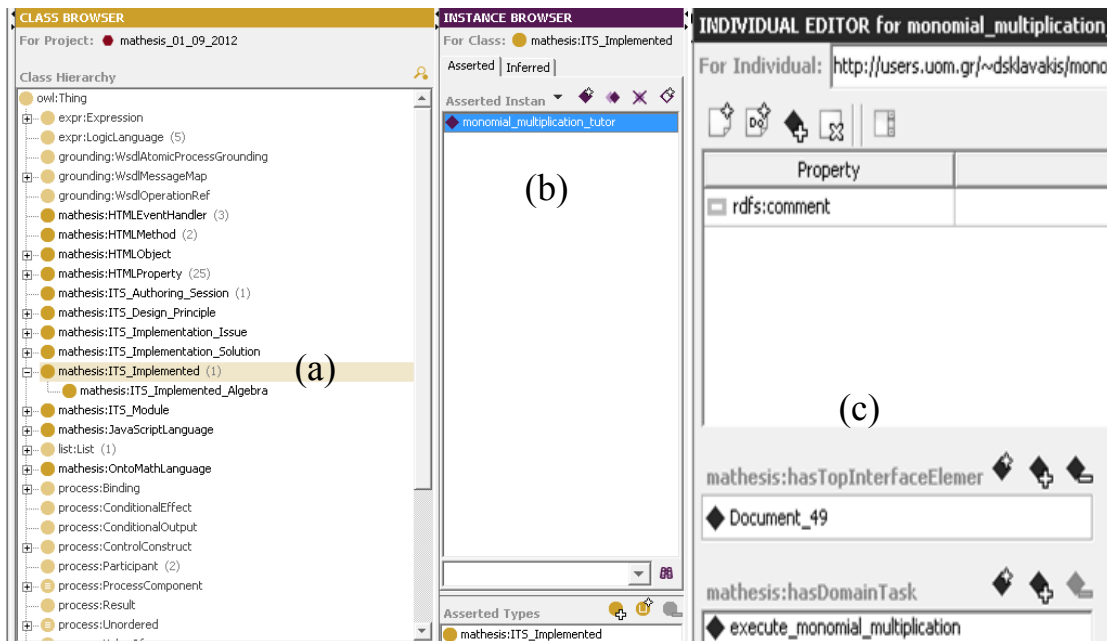



Fig. 4.4. (a) The ITS_Implemented hierarchy (b) Instance monomial-multiplication-tutor is selected (c) Properties of the selected instance

4.3 COGNITIVE MODEL INITIALIZATION

The next authoring action is to define the domain task that the tutor teaches, e.g., monomial multiplication. The author can add in the ontology new tasks using the “Create New Cognitive Task” button  (Figure 4.1a). The author is prompted to enter the cognitive task’s name (Figure 4.5).

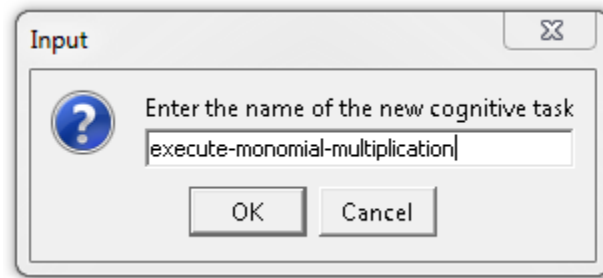



Fig. 4.5. Author is prompted by the tools to enter the name of a newly created cognitive task instance.

Authors can also select existing cognitive tasks to edit them, using the “Select Existing Cognitive Task” button  (Figure 4.1a). The created or selected instance of the domain task is added to property `hasDomainTask` that keeps a list of the tasks (here `execute-monomial-multiplication`) that the tutor teaches (Domain-Task instances). Figure 4.4c shows the domain task instance `execute-monomial-multiplication` listed as a value of the `hasDomainTask` property. The author fleshes out the domain tasks by executing the authoring processes developed by the meta-author according to each domain task as it will be explained later. As with the tutor instances, domain task instances are classified in a hierarchy with `Domain-Task` being the root (Figure 4.6). Notice that task `execute-monomial-multiplication` is actually an instance of the `Algebraic-Operation-Task` subclass.

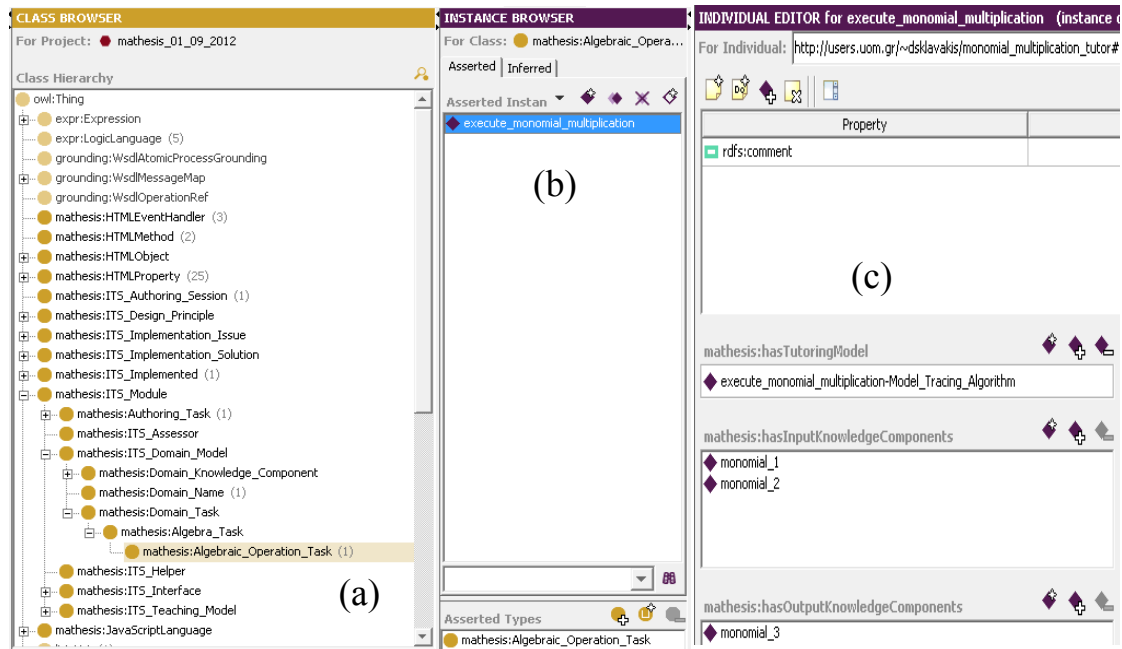


Fig. 4.6. (a) The Domain_Task hierarchy (b) Instance execute-monomial-multiplication is selected (c) Properties of the selected instance

For each domain task, the author must define in the ontology the tutoring model to be used, as well as the domain concepts given for the task and the domain concepts asked for the task. These authoring steps are performed by executing the corresponding authoring itprocesses created by the meta-author as it will be exemplified in section 4.7.

Instances of tutoring models are model-tracing, example-tracing, constraint-based and multiple-choice. These instances are initially just placeholders being able to represent any tutoring model; however, these instances turn from simple placeholders to specific implementations of the tutoring model they stand for. The meta-authoring model allows domain authors to associate each domain task with (possibly) a different teaching model. Furthermore, it allows using variations of the same tutoring model. For example, it could allow the domain author to add to the model-tracing teaching algorithm a service for

presenting solved examples of the tutored domain task. This was done to increase the model's flexibility. Of course flexibility increases complexity for the domain author who has to decide what tutoring model to choose for each domain task. This flexibility vs. complexity trade-off can be balanced using a default tutoring model. The meta-authoring model currently supports a single tutoring model, model-tracing, and the only flexibility provided to domain authors is to add or remove specific services. Tutoring model instances are classified in a hierarchy which has class ITS-Teaching-Model on top (Figure 4.7). In the ontology, the tutoring model of the domain task is kept by property `hasTutoringModel` of the domain task instance as shown in Figures 4.2 and 4.6c.

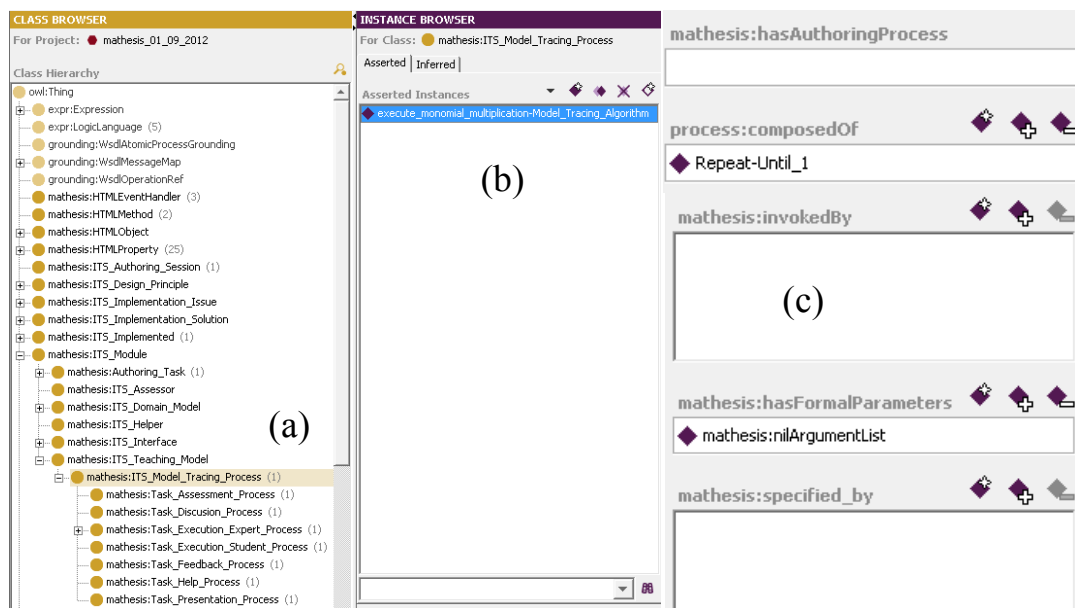


Fig. 4.7. (a) The ITS-Teaching-Model hierarchy. (b) Instance `execute-monomial-multiplication-Model-Tracing-Algorithm` has been selected. (c) Properties of the selected instance are shown.

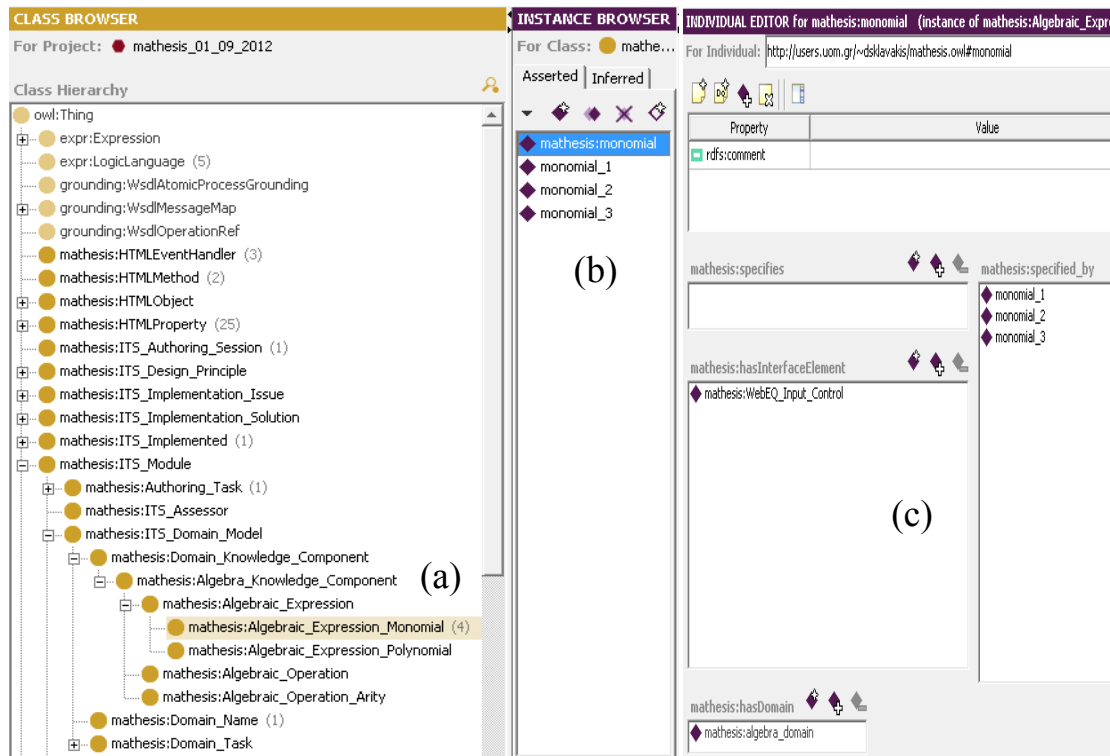


Fig. 4.8. (a) The Domain-Knowledge-Component hierarchy. (b) Instance monomial has been selected. (c) Properties of the selected instance are shown.

For the execute-monomial-multiplication task the input knowledge components are two monomials and the output knowledge component is their product, a monomial too. Domain concepts' instances are classified in a hierarchy with class Domain-Knowledge-Component on top (Figure 4.8a). In the ontology, the given and asked domain concepts of the domain task are kept by properties `hasInputKnowledgeComponents` and `hasOutputKnowledgeComponents` respectively (Figures 4.2 and 4.4c). Notice that the domain concept monomial is actually an instance of the Algebraic-Expression-Monomial subclass. In Figure 4.8b there are four instances: monomial, monomial-1, monomial-2 and monomial-3. Instance monomial is a *generic instance* created by the meta-author. When an author needs to define in a tutor the domain concept of monomial he/she has to clone the *generic*

instance monomial. The author creates these clones by executing the appropriate authoring processes created by the meta-author. Therefore, the state of the ontology shown in Figure 4.8 is after the author has executed the appropriate authoring processes that guided him to create the three specific instances, monomial-1, monomial-2 and monomial-3, by cloning monomial.

4.4 TUTORING MODEL INITIALIZATION

The top level representation of the tutor's procedural knowledge in the ontology is the model-tracing algorithm represented as a *generic composite tutoring process*, named ModelTracingAlgorithm. This algorithm is an implementation of the two-loop structure of intelligent tutoring systems described in (VanLehn, 2006) and was developed by the meta-author using the advanced Tutor Authoring Tools (Figure 4.2b). The tools allow meta-authors and advanced domain authors to create parts of the tutor's procedural knowledge (tutoring processes) directly, without executing authoring processes. This ability is necessary for complicated tutoring processes, like the ModelTracingAlgorithm, which demand high expertise and must be provided to non-expert authors as libraries, that is, *generic tutoring processes* in the MATHESIS framework terminology. When the non-expert author selects the ModelTracingAlgorithm as the tutoring model, the Tutor Initialization Authoring Tools (Figure 4.1a) copy its structure and create a new instance for the authored task, `execute_monomial_multiplication-Model_Tracing_Algorithm` (Figure 4.1c). Consequently, the meta-author can define any number of framework-specific, generic tutoring models (e.g. model-tracing, example-tracing, constrained-based, other) ready to be selected by the non-expert author and adapted by the execution of authoring processes for the authored domain task.

The tree structure of the process, adapted and displayed by the Tutor Initialization Authoring tools for the `execute-monomial-multiplication` task, is shown in Figure 4.1c. Each step of the algorithm is a top level tutorial action:

1. Loop over problems with the same domain task, e.g., monomial multiplication, until the assessment subsystem signifies that the task is adequately acquainted by the student (`Repeat-Until_1(tutoring-done)`). Instance `tutoring-done` is a MATHESIS *generic predicate* created by the meta-author. The implementation of this predicate in the target language is performed by executing authoring processes.
2. Present, for each problem, the initial problem solving state (`Perform_6 (execute-monomial-multiplication-Presentation())`).
3. Loop over the solution steps for a specific problem that teaches the domain skill, e.g. $2x^2y \cdot 3xy^3$ (`Repeat-Until_8(task_is_achieved)`).
4. Get the correct solution(s) from the domain expertise model (`Perform_13 (execute-monomial-multiplication-Execution())`).
5. Provide help/hint (`Perform_17 (execute-monomial-multiplication-Help())`).
6. Get the student solution (`Perform_21 (execute-monomial-multiplication-Execution-Student())`).
7. Provide feedback to the student (`Perform_25 (execute-monomial-multiplication-Feedback())`).
8. Assess student solution (`Perform_29(execute-monomial-multiplication-Assessment())`).
9. Discuss student solution (`Perform_33 (execute-monomial-multiplication-Discussion())`).

Each of these steps is also a *composite tutoring process* that analyses the tutorial steps further down to more simple ones like giving a hint, showing an example, recalling math formulas or rules and so on. In programming terms, the `ModelTracingAlgorithm` composite tutoring process, when translated to code, constitutes the main function that controls the whole tutoring process by calling other functions. This recursive analysis of

the tutoring steps ends when a composite tutoring process contains only atomic processes corresponding to simple statements.

For example, in the case of the `execute-monomial-multiplication` task, the `execute-monomial-multiplication-Execution` process (Figure 4.1c, `Perform_13`) is analyzed in two other composite processes: `multiplyCoefficients` and `multiplyMainParts`. These two processes form the tutor's *domain expertise model*, which calculates the correct answer(s) in each step in order to be compared against the student's answer. Figure 4.9 shows part of the structure of process `multiplyMainParts`. Once again, there are two options for the meta-author on how to create these processes from the side of the non-expert author:

- i. The meta-author must develop the authoring processes that, when executed, guide the non-expert author in a step-by-step manner to implement them, and
- ii. the meta-author creates these tutoring processes directly, using the advanced Tutor Authoring Tools (Figure 4.1b), and then develops simpler authoring processes that just guide the non-expert author in selecting the former from the MATHESIS ontology.

The first option entails considerable workload for the meta-author but it is closer to the principles and objectives of the MATHESIS framework, which aims at the representation of authoring and tutoring knowledge at the greatest possible detail. The second option is easier for the meta-author but hides and obscures the authoring knowledge that was actually used to develop these tutoring processes.

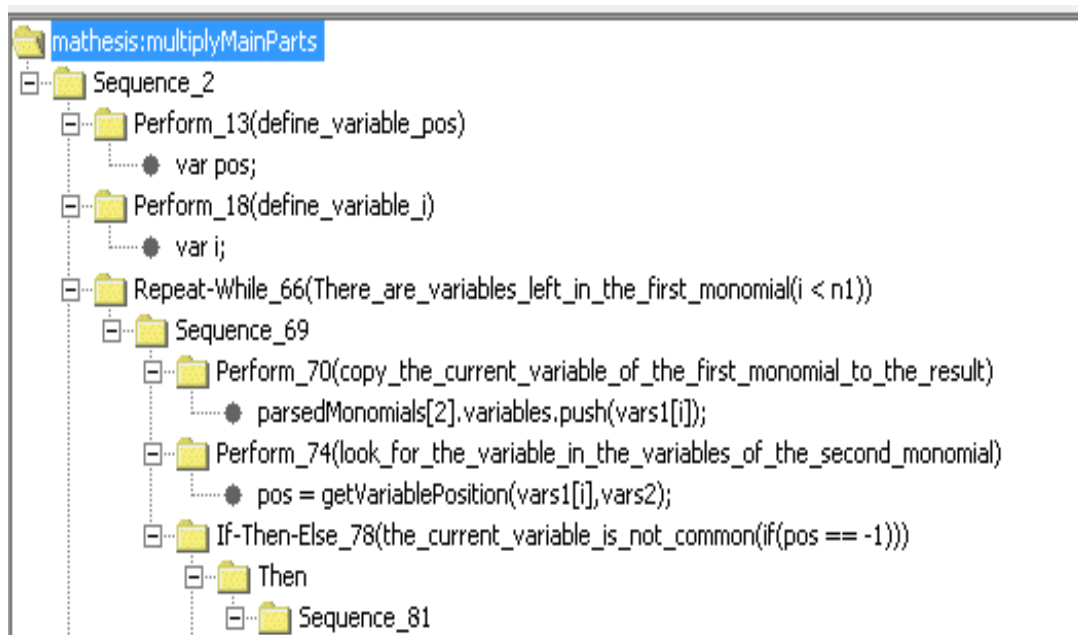


Fig. 4.9. Representation of the JavaScript function multiplyMainParts

4.5 PROGRAM CODE MODEL

Each JavaScript statement is represented by an instance of the JavaScriptStatement class, a subclass of AtomicProcess (Figure 4.10). Following the OWL-S representational scheme, these instances are parameters to Perform constructs. The JavaScriptStatement class has subclasses which classify the JavaScript statements in various classes such as DefineVariable, InitializeVariable, AssignValueToVariable, InvokeFunction, InvokeMethod, SetProperty. Each subclass has properties that represent the various parts of the corresponding JavaScript statements. For example, the InvokeFunction class has three properties, hasInvokedFunction, hasArgumentsList and hasAssignedVariable.

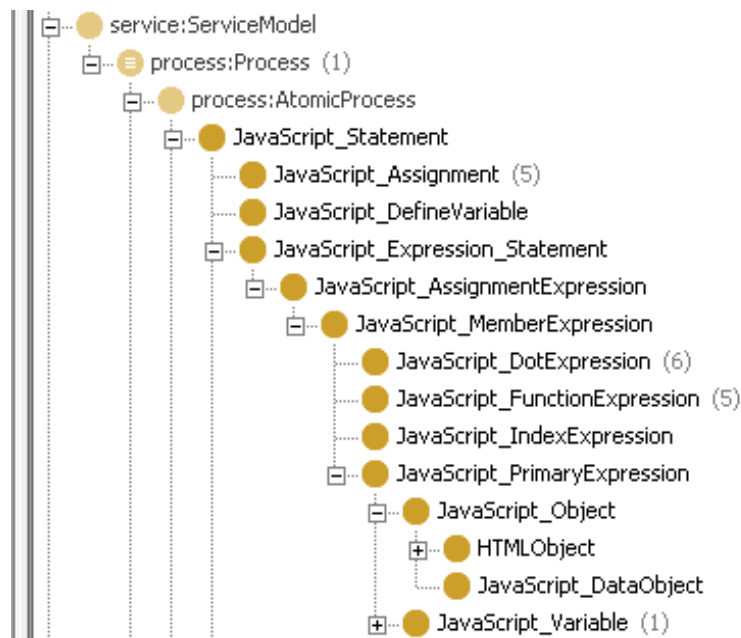


Fig. 4.10 Part of the JavaScript_Statement ontology

From the values of these properties, the authoring tools create and display the actual JavaScript code as shown in Figure 4.9 (the small disks under the Perform constructs). Such a detailed model of the JavaScript language allows the authoring processes to guide the non-expert author in building the tutor's code by selecting the appropriate JavaScriptStatement subclass and the values of the related properties. Therefore, a non-expert author does not need to know the JavaScript syntax but only needs to have some general programming knowledge. As far as it concerns the semantic validation of the produced JavaScript code, the tools do not provide any special assistance to the authors. That is, the produced JavaScript code is syntactically correct, however whether this code exhibits the intended behavior is a matter of correct design and analysis of the authoring processes.

4.6 INTERFACE MODEL INITIALIZATION

The tutor's author must also create in the MATHESIS ontology the user interface model. In the current implementation the interface is a web page and therefore the user interface model is a representation of the HTML Document Object Model (DOM). The tutor's placeholder for the interface model is kept by property `hasTopInterfaceElement` that holds the root element of the user interface, a Document instance (Figures 4.2 & 4.4c). An ontological representation of the HTML elements with their properties and values has been developed. When the author creates the tutor's instance for the first time, the Tutor Initialization Authoring Tools automatically create the ontological representation of an empty HTML page. The representation of the HTML code and the corresponding DOM of the user interface for the monomial multiplication tutor are shown in Figure 4.11.

The instances surrounded by the dotted line (`Document_49`, `Html_51`, `Head_53` and `Body_54`) were created automatically by the Tutor Initialization Authoring Tools, while the rest were created by the execution of authoring processes. Each object defined in the HTML code is represented as an instance of the corresponding `HTMLObject` subclass (`Document`, `Html`, `Head`, `Body`, `Applet`, `Web_Eq_Input_Control`). Each `HTMLObject` instance has the corresponding HTML properties, like the *id* property, pointed by property `hasHTMLProperty` (instances `Web-Eq-Input-Control-1-id`, `Web-Eq-Input-Control-2-id`). The HTML values of these properties are represented by their corresponding ontological properties (*html-property-value* = "expressionInputControl"). The DOM tree structure is represented via two properties, `hasFirstChild` and `hasNextSibling`.

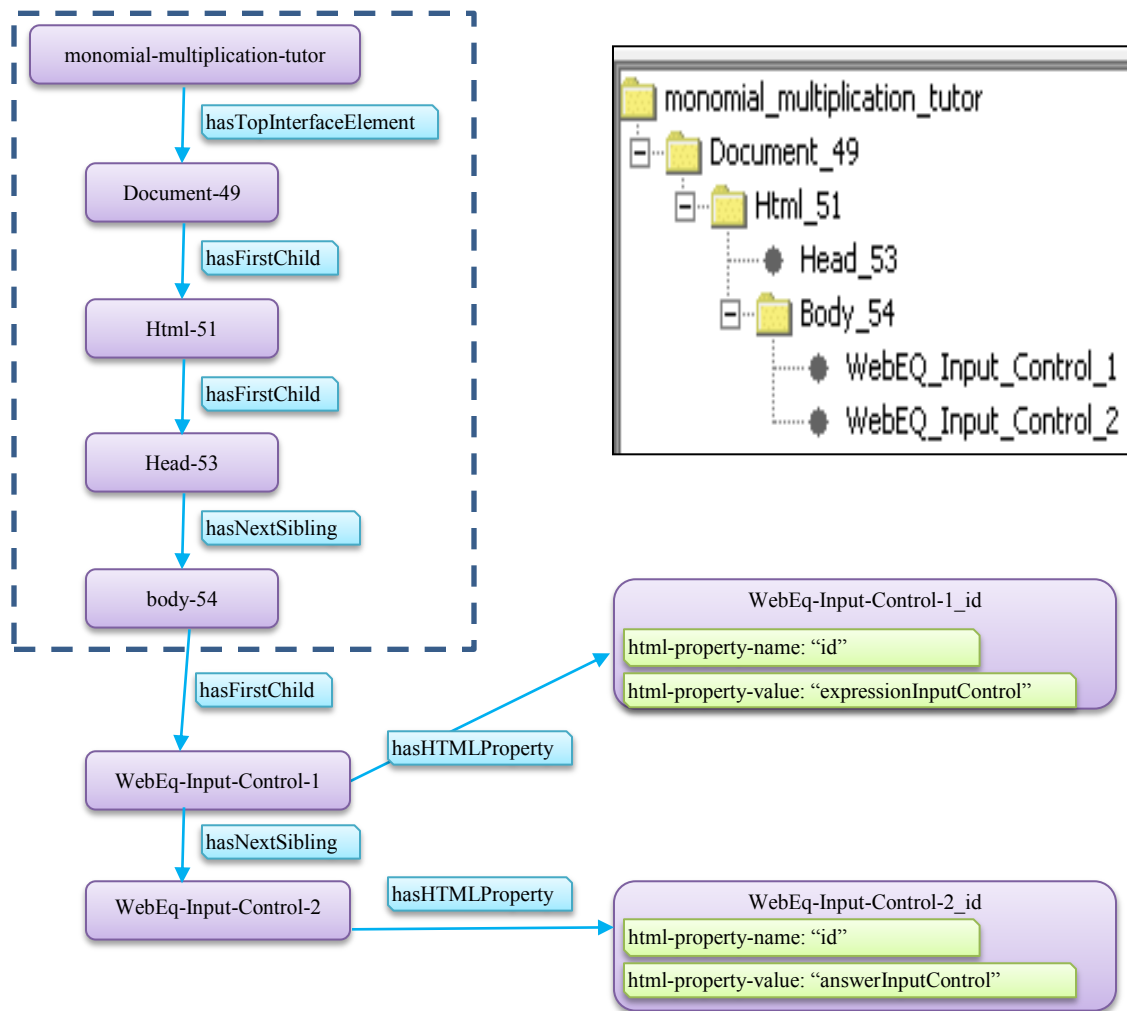


Fig. 4.11. The HTML User Interface DOM Ontological (left) and Visual (right, top) Representation

This representation allows for bi-directional creation of the HTML part of the user interface:

- i. The author, either guided by the authoring processes or using the Tutor Authoring Tools, creates within the MATHESIS ontology the representation of the DOM tree and then, by traversing the ontology, the translation tools generate the corresponding HTML code (top-down), or

- ii. The user interface is created using any Web-page authoring program and then the HTML file is parsed by Java's XML parser creating a DOM structure, which in turn is transformed into its corresponding ontological representation for further authoring by the authoring tools (bottom-up).

For different interface implementations like Java or Flash, appropriate ontological representations and translation programs must be developed.

Instances `WebEq-Input-Control-1` and `WebEq-Input-Control-2` were created by the author. The first one is the interface element that is used to present the two monomials to be multiplied and represents the algebraic expression area of the MATHESIS Algebra Tutor (Figure 2.1, bottom left). The second one is used for the student answering area of the MATHESIS Algebra Tutor (Figure 2.1, bottom right). The meta-author associated monomial instances with the WebEq Input Control interface elements by setting property `has-Interface-Elements` of the generic monomial instance to `WebEQ-Input-Control` (Figure 4.8c). Then, the meta author created two authoring process called `define-interface-elements-for-input-knowledge-components` and `define-interface-elements-for-output-knowledge-components` correspondingly that guide the author in creating the two WebEq instances and naming them "expressionInputControl" and "answerInputControl" correspondingly (Figure 4.11, bottom right).

4.7 EXECUTION OF AUTHORING PROCESSES

In the previous paragraphs the initial, simple, top-level authoring tasks were described. As these constitute *authoring expertise* they should be carried out from the author by executing the corresponding *authoring processes* developed by the meta-author. However, as these authoring tasks are initial, simple and top-level, it was decided

to implement them as tools in the Tutor Authoring Tools window (Figure 4.1a); alternatively, they could have been implemented as authoring processes.

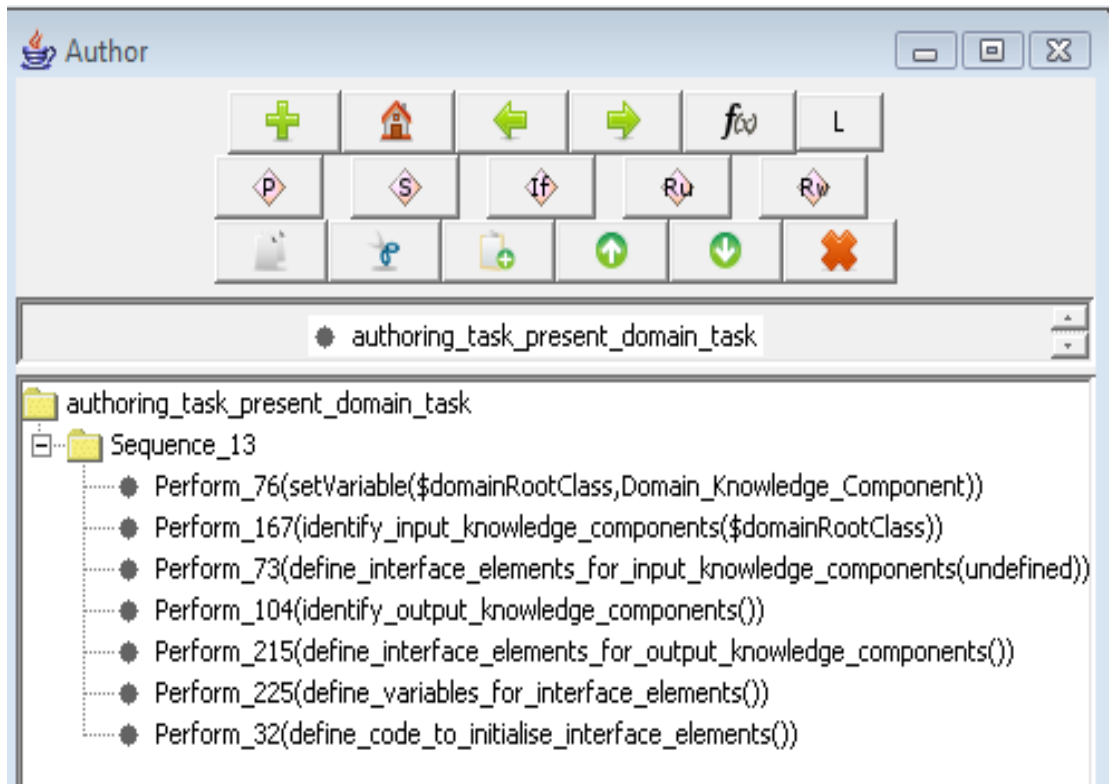


Fig. 4.12. The Authoring Processes Authoring (Meta-Authoring) Tools displaying Authoring Process `authoring_task_present_domain_task`.

In the MATHESIS framework, authoring processes capture the authoring expertise of expert authors and make it available to non-expert authors. They form an executable model of the expert authors' expertise. When executed, authoring processes guide non-expert authors to develop their own tutors. Meta-authors develop authoring processes using the Authoring Processes Authoring Tools (Meta-Authoring Tools) (Figure 4.12). These tools allow creating, editing and executing authoring processes²¹.

²¹ http://ai.uom.gr/dsklavakis/en/mathesis/kes2011/02-Authoring_Processes.mp4

For example, each tutoring process of the `execute_monomial_multiplication-Model_Tracing_Algorithm` (Figure 4.1c) has a corresponding authoring process. The code for tutoring process `execute-monomial-multiplication-Presentation` (Figure 4.1c, `Perform_6`) is created by the execution of the corresponding authoring process `authoring-task-present-domain-task` shown in Figure 4.12. The author must perform the following authoring sub-tasks:

1. Identify the input (given) domain concepts of the domain task. For the domain task of monomial multiplication these are the two monomials to multiply. This authoring task demands authoring knowledge of cognitive task analysis. This authoring knowledge is encoded by the meta-author in authoring process `identify-input-knowledge-components` (Figure 4.12, `Perform_167`).
2. Define the interface elements that present the given math concepts (monomials) identified in the previous step. In our implementation of the tutor this is a `WebEq_Input_Control` applet, `WebEq_Input_Control_1` (see Figure 4.11). This authoring task demands authoring knowledge of interface design and HTML programming. This authoring knowledge is encoded by the meta-author in authoring process `define-interface-elements-for-input-knowledge-components` (Figure 4.12, `Perform_73`).
3. Repeat the aforementioned steps for the task's output (asked) concepts, that is, a monomial holding the product. The corresponding interface element is another `WebEq_Input_Control` applet, `WebEq_Input_Control_2` (see Figure 4.11). This authoring knowledge is encoded by the meta-author in authoring processes `identify-output-knowledge-components` (Figure 4.12, `Perform_104`) and `define-interface-elements-for-output-knowledge-components` (Figure 4.12, `Perform_215`).
4. Define the JavaScript variables that hold the references of the `WebEq_Input_Control` applets. This authoring task demands authoring knowledge of JavaScript

- programming. This authoring knowledge is encoded by the meta-author in authoring process `define-variables-for-interface-elements` (Figure 4.12, Perform_225).
5. Define the JavaScript code that initializes the variables referencing the `WebEq_Input_Control` applets. This authoring task also demands authoring knowledge of JavaScript programming. This authoring knowledge is encoded by the meta-author in authoring process `define-code-to-initialise-interface-elements` (Figure 4.12, Perform_32).

These are top-level authoring tasks. The ONTOMATH language allows the analysis of these tasks in any level of detail, that is, in authoring tasks of any granularity. To illustrate this key point, the execution of the authoring processes that perform tasks 1,2, 4 and 5 are describe in more technical detail. Programming-savvy readers can follow the ONTOMATH code using the definitions of the ONTOMATH statements given in Table 3.2 as well as in Appendix B.

The execution of these authoring processes starts from process `authoring-task-present-domain-task` (Figure 4.12). Statement `Perform_76(setVariable($domainRootClass, Domain_Knowledge_Component))` sets the value of variable `$domainRootClass` to `Domain_Knowledge_Component`. This is the root class that contains the math domain concepts (Figure 4.8a). Statement `Perform_167(identify-input-knowledge-components($domainRootClass))` calls process `identify_input_knowledge_components` (Figure 4.13) with `$domainRootClass` as a parameter. This process guides the author to find the input math concept(s) for the task of monomial multiplication, i.e. two monomials. It is executed as follows:

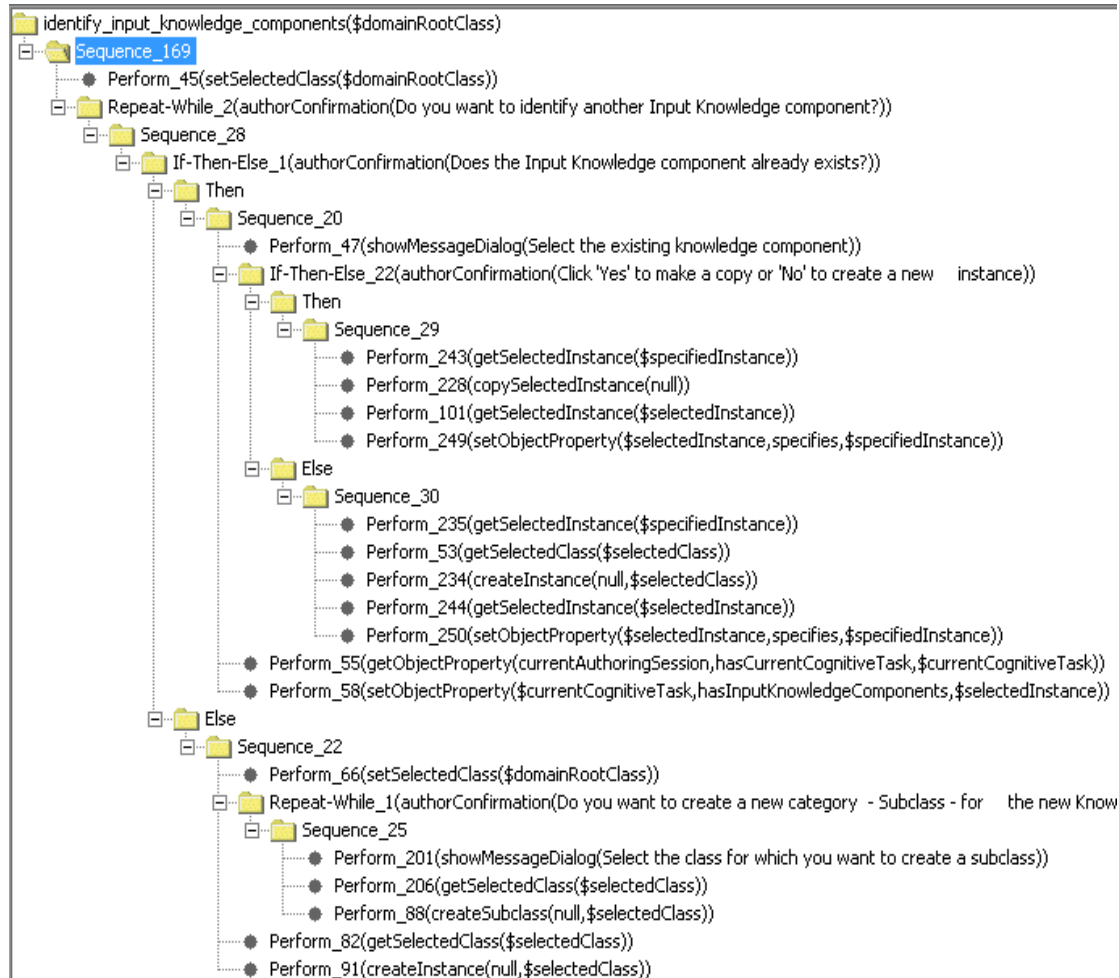


Fig. 4.13. The identify_input_knowledge_components authoring process.

1. Perform_45(setSelectedClass(\$domainRootClass)) sets the focus on class Domain-Knowledge-Component in the CLASS BROWSER panel of the MATHESIS ontology tab (Figure 4.14). Repeat-While_2(authorConfirmation(Do you want to identify another input knowledge component?)) starts an iteration for each input knowledge component that the author must identify. The condition, authorConfirmation, is an OntoMath predicate that displays a

- Yes/No question to the author and evaluates to True/False according to the author's answer. The author must look into the subclasses of class `Domain_Knowledge_Component` to find whether the input knowledge component, in our example monomial, already exists in the ontology or not. The author is asked about that fact by statement `If-Then-Else_1(authorConfirmation(Does the input knowledge component already exists?))`. Since the knowledge component, monomial, exists (Figure 4.14), the author answers "Yes" and `Sequence_20` is executed. If the monomial input knowledge component does not exist then `Sequence_22` is executed. `Repeat-While_1(authorConfirmation(Do you want to create a new category – Subclass – for the new Knowledge Component?))` initiates a loop (`Sequence-25`) that guides the author in creating new subclass(es) of the `Domain-Knowledge-Component` class. When the author creates the last subclass of the hierarchy, `Perform_91(createInstance(null, $selectedClass))` creates a new instance and asks the author for its name (e.g. `monomial_1`).
- `Perform_47(showMessageDialog(Select the existing knowledge component))` prompts the author to select monomial in the `INSTANCE BROWSER` panel of the `MATHESIS` ontology tab (Figure 4.14). Statement `If-Then-Else_22(authorConfirmation(Click 'Yes' to make a copy or 'No' to create a new instance))` asks the author whether he/she wants to copy the existing monomial instance or create a new one. In Protégé, copying an instance also copies its properties and their values, while creating a new instance sets property values to null. In our example the author wants to keep the default values of the generic monomial instance and therefore `Sequence_29` will be executed.
 - `Perform_243(getSelectedInstance($specifiedInstance))` gets the selected monomial instance from the `INSTANCE BROWSER` panel of the `MATHESIS` ontology tab (Figure 4.14) and sets variable `$specifiedInstance` to hold a reference to it.

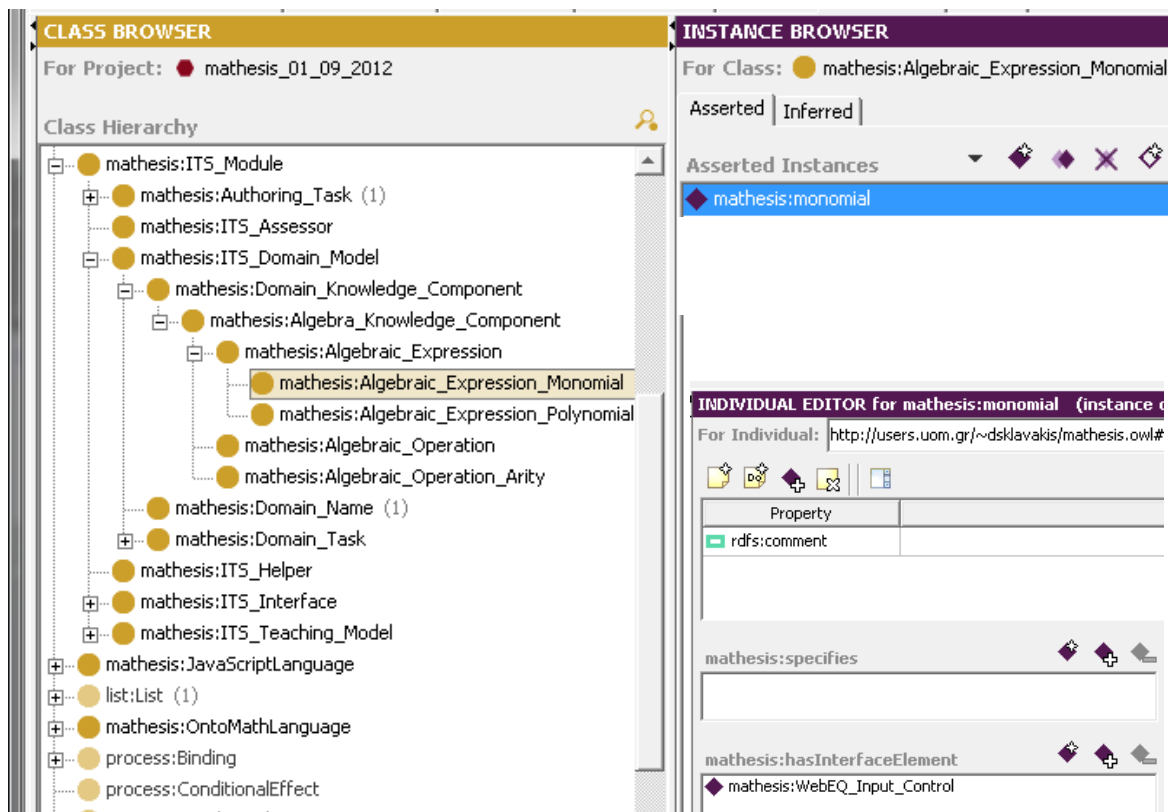


Fig. 4.14. Locating a monomial instance in the ontology

4. Perform₂₂₈(copySelectedInstance(null)) gets the selected monomial instance from the INSTANCE BROWSER panel of the MATHESIS ontology tab (Figure 4.14) and creates a new copy of it. The parameter to this OntoMath statement is the name of the newly created instance. If it is null, as in our example, the author is prompted for the name. In our example, the author named the new monomial instance monomial₁ (Figure 4.15).
5. Perform₂₄₉(setObjectProperty(\$selectedInstance, specifies, \$specifiedInstance)) sets the value of property specifies of the newly created monomial₁ (\$selectedInstance) to monomial (\$specifiedInstance). Property specifies of a knowledge component (mathematical concept), points to the generic knowledge component instance from which a

specific knowledge component was generated. In our example the semantics is that `monomial_1` is a specification of `monomial` (Figure 4.15).

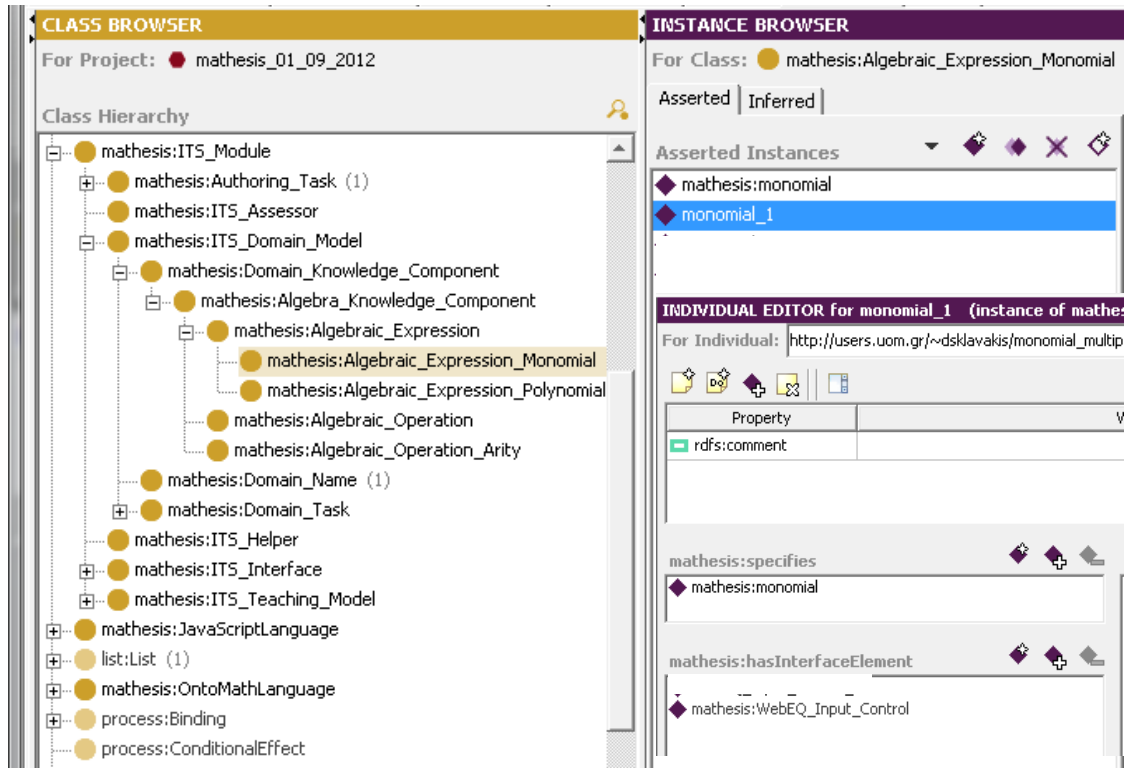


Fig. 4.15. Creating a new instance of monomial

6. `Perform_55(getObjectProperty(currentAuthoringSession, hasCurrentCognitiveTask, $currentCognitiveTask))` gets the value of property `hasCurrentCognitiveTask` of instance `currentAuthoringSession`, i.e. `execute-monomial-multiplication`, and assigns it to variable `$currentCognitiveTask`. Instance `currentAuthoringSession` (Figure 4.16) keeps information about the current authoring session like which is the tutor currently being authored, what is the cognitive task currently being authored etc.
7. `Perform_58(setObjectProperty($currentCognitiveTask, hasInputKnowledgeComponents, $selectedInstance))` sets the value of property `hasInputKnowledgeComponents` of instance

execute-monomial-multiplication (\$currentCognitiveTask) to monomial_1 (\$selectedInstance)

(Figure 4.16).

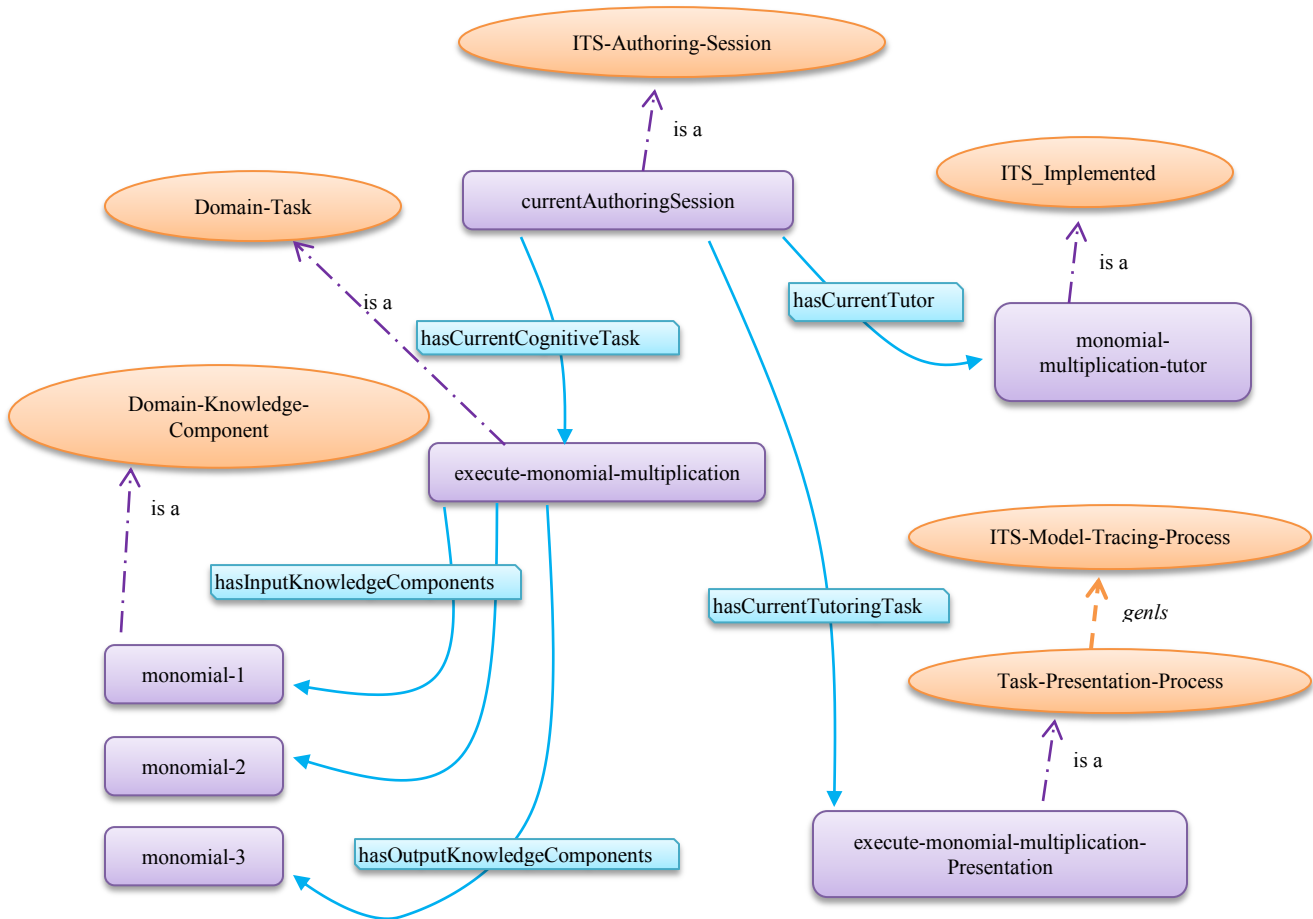


Fig. 4.16 Instance currentAuthoringSession for the monomial-multiplication-tutor

Up to this point the author has been guided in creating and defining instance monomial_1 as an input math concept of the monomial multiplication task. Execution control returns to Repeat-While_2 and the author is asked if he/she wants to define another input component. By answering ‘Yes’, the author creates the second input math concept , monomial_2. The author terminates the execution of this loop when he/she thinks all input

knowledge components have been defined. That also terminates execution of authoring process `identify_input_knowledge_components` and control is returned back to the calling process, `authoring-task-present-domain-task` (Figure 4.12). This one calls in turn authoring process `define-interface-elements-for-input-knowledge-components` (Figure 4.17).

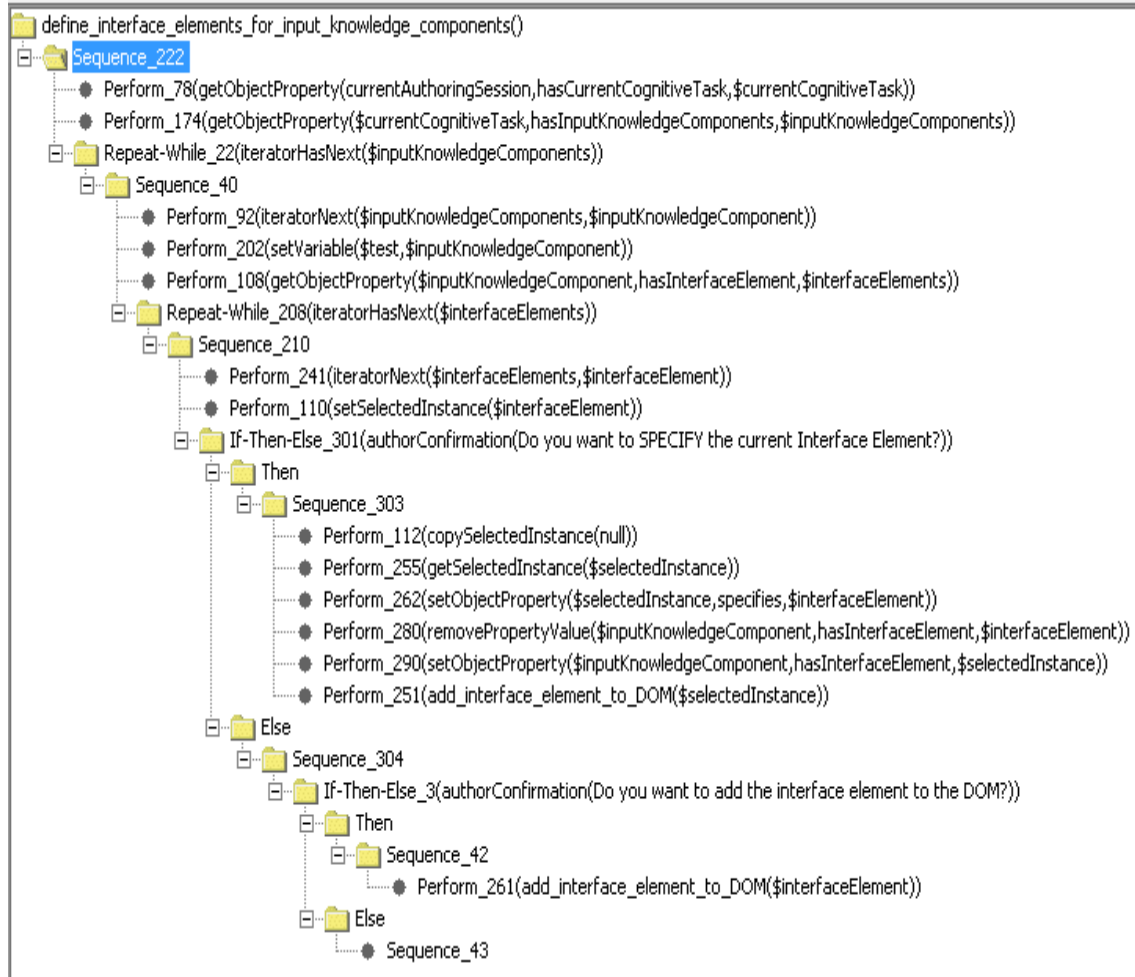


Fig. 4.17. The `define-interface-elements-for-input-knowledge-components` authoring process

This process guides the author in defining the HTML interface elements that will present the input knowledge component(s) defined previously, here `monomial_1` and `monomial_2`. The execution of this process takes the following steps:

1. `Perform_78(getObjectProperty(currentAuthoringSession, hasCurrentCognitiveTask, $currentCognitiveTask))` gets the value of property `hasCurrentCognitiveTask` of instance `currentAuthoringSession`, i.e. `execute_monomial_multiplication`, and assigns it to variable `$currentCognitiveTask` (Figure 4.16).
2. `Perform_174(getObjectProperty($currentCognitiveTask, hasInputKnowledgeComponents, $inputKnowledgeComponents))` gets the value of property `hasInputKnowledgeComponents` of instance `execute_monomial_multiplication` (pointed by `$currentCognitiveTask`) and assigns it to variable `$inputKnowledgeComponents`. This value is the list of the input knowledge components of the monomial multiplication task. In our example, these are `monomial_1` and `monomial_2` (Figure 4.16). Hence, `$inputKnowledgeComponents = {monomial_1, monomial_2}`.
3. `Repeat-While_22(iteratorHasNext($inputKnowledgeComponents))` loops over the list of the `$inputKnowledgeComponents = {monomial_1, monomial_2}`.
4. `Perform_92(iteratorNext($inputKnowledgeComponents,$inputKnowledgeComponent)` gets the next input knowledge component, here `monomial_1`.
5. `Perform_108(getObjectProperty($inputKnowledgeComponent, hasInterfaceElement, $interfaceElements)` gets the list of the HTML interface elements (`$interfaceElements`) that can be used to represent `monomial_1` (`$inputKnowledgeComponent`), here `WebEQ-Input-Control` (Figure 4.15). This list has been inherited from the generic monomial instance when it was copied to give `monomial_1`. Of course, the values of this list (`WebEQ-Input-Control`) had been defined beforehand by an expert author (meta-author). This represents a good example of lowering the expertise threshold via expert knowledge reuse. The `WebEQ` input control is a highly sophisticated Java applet for displaying and editing mathematical expressions. The knowledge of its existence and usage is definitely top level expertise.

6. Repeat-While_208(iteratorHasNext(\$interfaceElements)) loops over the list of the proposed HTML interface elements. In our example there is only one, WebEQ-Input-Control.
7. Perform_110(setSelectedInstance(\$interfaceElement)) sets the focus on WebEQ-Input-Control, in order to be specified into a new instance, Applet-WebEQ-Input-Control-1, just like monomial was specified to monomial_1.
8. The OntoMath statements of Sequence_303 guide the author in creating the new instance, Applet-WebEQ-Input-Control-1, and replace WebEQ-Input-Control with Applet-WebEQ-Input-Control-1 in the hasInterfaceElements list of monomial_1.

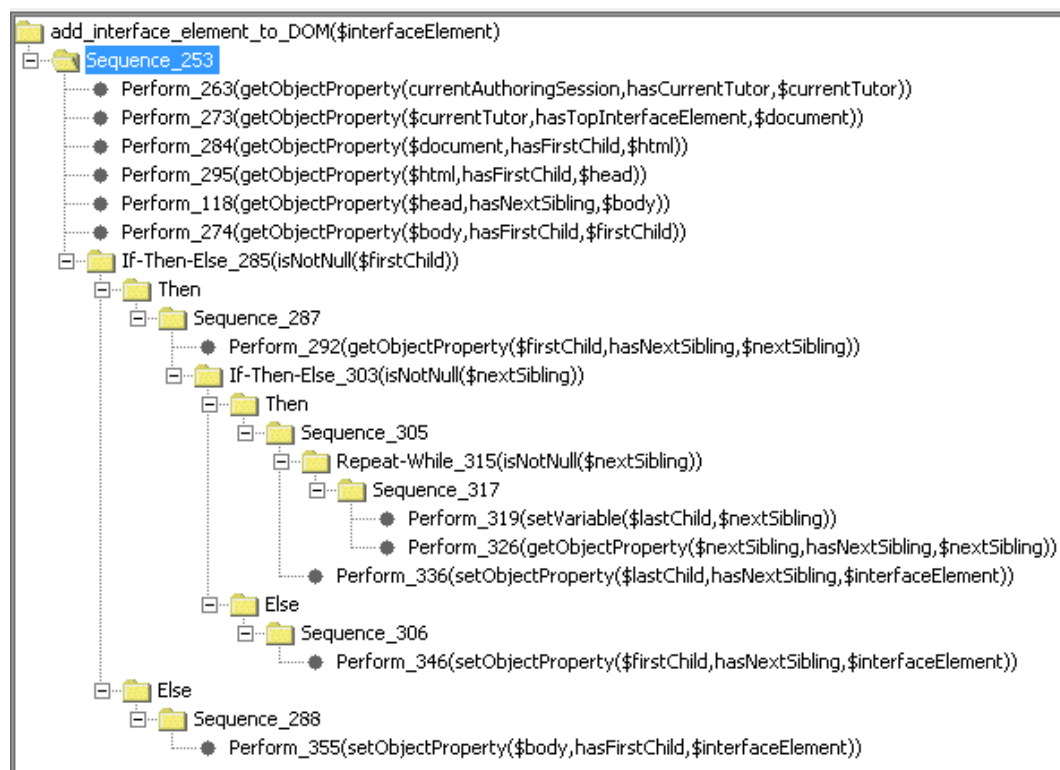


Fig. 4.18 The add-interface-element-to-DOM authoring process

9. Finally, process add-interface-element-to-DOM (Figure 4.18) is called to add the newly created instance Applet-WebEQ-Input-Control-1 in the ontological representation of the

HTML interface's Document Object Model (Figure 4.11, dotted rectangle). The statements of Sequence_253 travel through the ontological representation of the DOM tree to get to the Body element of the tutor's user interface, Body_54 (Figure 4.11, bottom of dotted rectangle). Perform_355(setObjectProperty(\$body, hasFirstChild, \$interfaceElement)) adds AppletWebEQ_Input_Control-1 (\$interfaceElement) as a child of Body_54 (Figure 4.11, below the dotted rectangle).

10. The execution of process add-interface-element-to-DOM is terminated and execution control returns back to process define-interface-elements-for-input-knowledge-components (Figure 4.17).
11. Repeat-While_208(iteratorHasNext(\$interfaceElements)) ends as there is not any interface element left for monomial_1.
12. Repeat-While_22(iteratorHasNext(\$inputKnowledgeComponents)) loops over the list of the \$inputKnowledgeComponents = {monomial_2}.
13. Steps 5 to 7 are repeated for monomial_2. However, the author now does not want to define a new interface element for monomial_2 since AppletWebEQ_Input_Control-1 will be used to present it. Therefore, steps 8 to 10 are not executed by the author. All Repeat-While loops are terminated and control returns back to process authoring-task-present-domain-task (Figure 4.12).

Processes identify-output-knowledge-components (Perform_104) and define-interface-elements-for-output-knowledge-components (Perform_215), perform the authoring actions described above but for the output knowledge components of the monomial multiplication task. These processes guide the author to create a new instance of monomial, monomial_3 that corresponds to the result of the multiplication and a new WebEQ-Input-Control instance, Applet-WebEQ-Input-Control-2, where the student enters his/her answer. The resulting ontological representation of the tutor, with its user interface is shown in Figure 4.19.

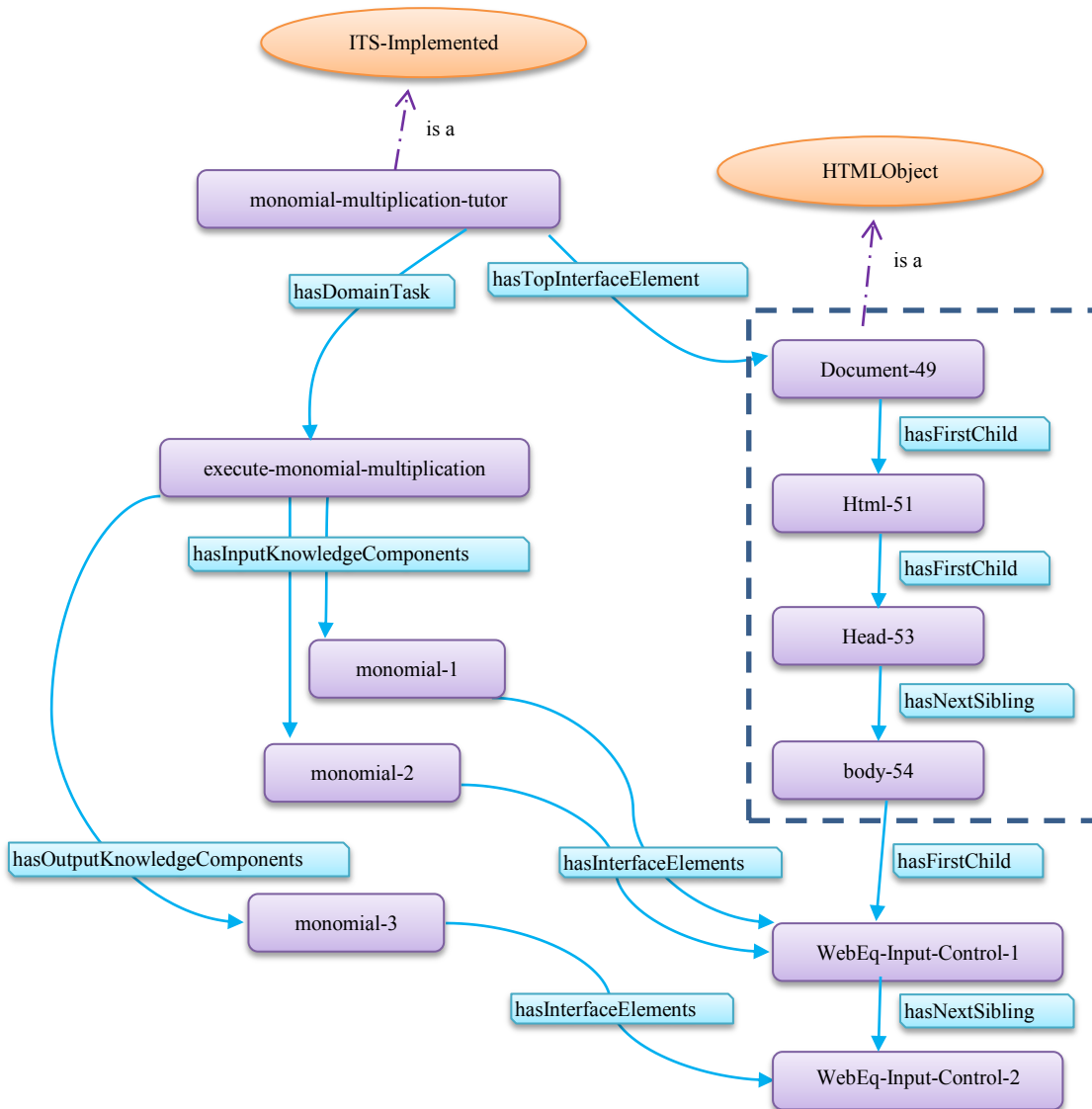


Fig. 4.19 Ontological representation of a monomial tutor with its user interface

The next authoring task is to define the JavaScript variables that will hold the references of the WebEq_Input_Control applets. This authoring task demands authoring knowledge of JavaScript programming. To perform this task process authoring-task-present-

domain-task calls authoring process `define-variables-for-interface-elements` (Figure 4.20). This process is executed as follows:

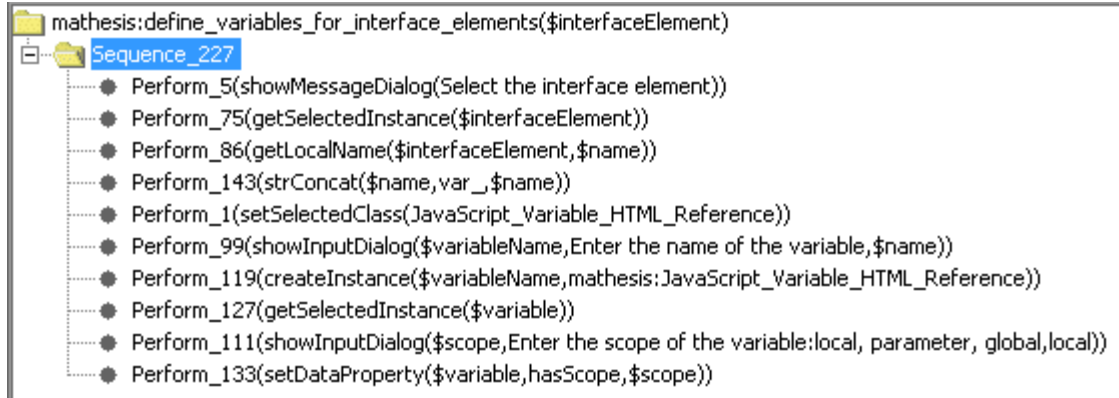


Fig. 4.20 Authoring process `define-variables-for-interface-elements`

1. `Perform_5(showMessageDialog(Select the interface element))` prompts the author to select the interface element for which the reference variable will be created, here `WebEq_Input_Control_1`.
2. `Perform_86(getLocalName($interfaceElement,$name))` makes the assignment `$name=WebEq_Input_Control_1`. `Perform_143(strConcat($name,var_,$name))` sets `$name` to `var_WebEq_Input_Control_1`.
3. `Perform_1(setSelectedClass(JavaScript_Variable_HTML_Reference))` sets the focus on class `JavaScript_Variable_HTML_Reference` in the CLASS BROWSER. This class contains the JavaScript variables that hold references to HTML interface elements (Figure 4.21, left).
4. `Perform_99(showInputDialog($variableName, Enter the name of the variable, $name))` prompts the author to input the name of the JavaScript variable that will be created and suggests as a default name `var_WebEq_Input_Control_1`. The author can give a different name, `expressionInputControl` in this example.

Perform_119(createInstance(\$variableName,mathesis:JavaScript-Variable-HTML-Reference)) creates the variable (Figure 4.21, center).

5. Perform_111(showInputDialog(\$scope,"Enter the scope of the variable: local, parameter, global", local)) prompts the author to enter the scope of the variable, *local*, *parameter* or *global* suggesting as default *local*. The author enters *global* and its input is stored in variable \$scope.
6. Perform_133(setDataProperty(\$variable, hasScope, \$scope) sets the hasScope property of the newly created variable expressionInputControl to *global* (Figure 4.21, right).

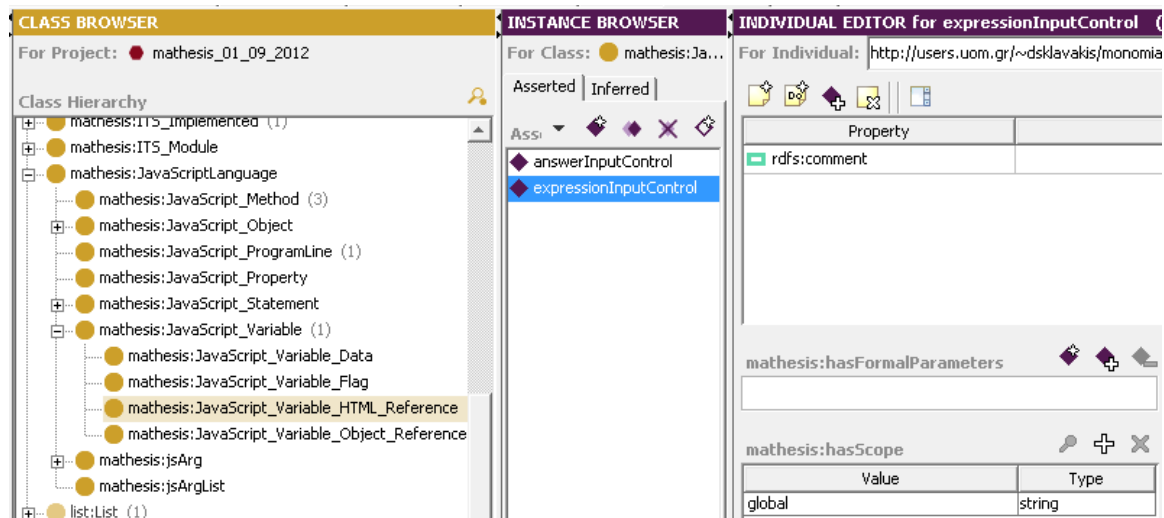


Fig. 4.21 The expressionInputControl JavaScript variable

By re-executing process define-variables-for-interface-elements the author also defines JavaScript variable answerInputControl that will reference applet WebEq_Input_Control_2. Now, the final authoring task is to define the JavaScript code that will initialize the newly created variables. This code is actually two JavaScript assignment statements. To write this code an author must have knowledge of JavaScript and HTML programming:

- a. First, the HTML *id* property of `WebEq_Input_Control_1` must be set to a value, e.g. “`expressionInputControl`”, and that of `WebEq_Input_Control_2` to another value, e.g. “`answerInputControl`”.
- b. Then the author must write the two JavaScript statements that assign to two variables references to the HTML elements `WebEq_Input_Control_1` and `WebEq_Input_Control_2` using the *getElementById* JavaScript function. These statements are `expressionInputControl = getElementById(“expressionInputControl”) and answerInputControl = getElementById(“answerInputControl”) correspondingly.`

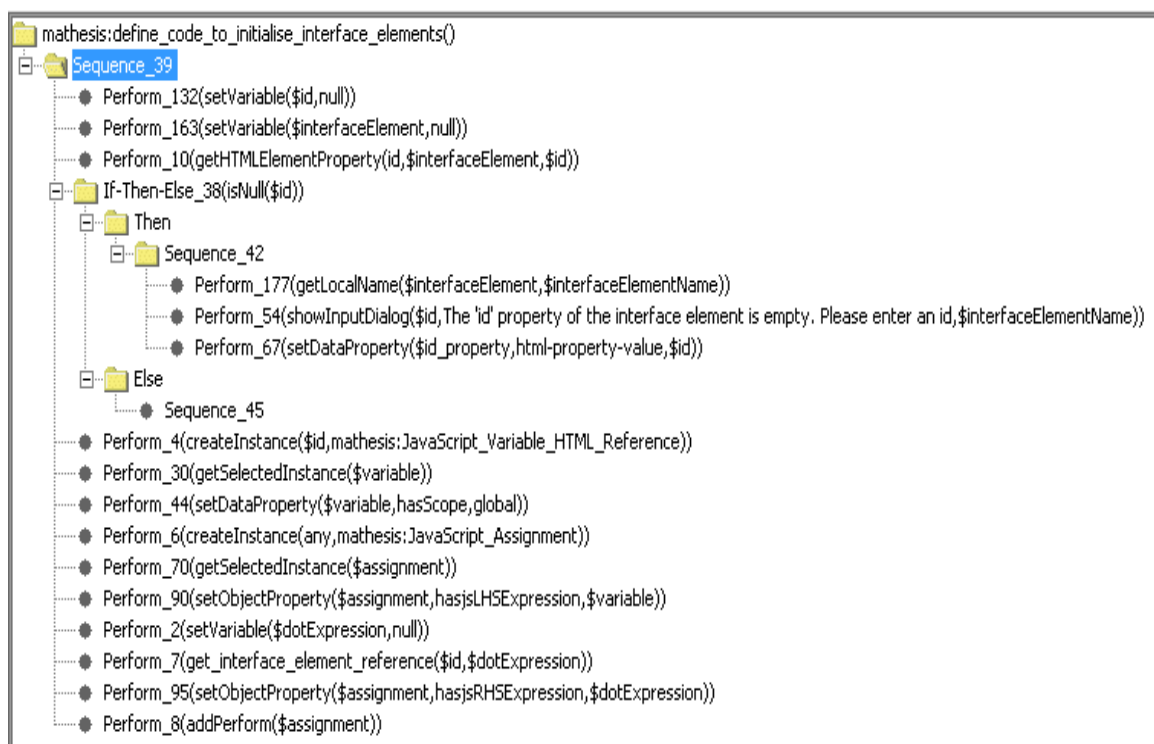


Fig. 4.22 The `define_code_to_initialize_interface_elements` authoring process

To perform these tasks, process `authoring-task-present-domain-task` calls authoring process `define_code_to_initialize_interface_elements` (Figure 4.22). This process is executed as follows:



Fig. 4.23 The getHTMLElementProperty authoring process

1. Perform_10(getHTMLElementProperty(id,\$interfaceElement,\$id) calls authoring process getHTMLElementProperty (Figure 4.23) with the following bindings: \$propertyName=id, \$htmlElement=null and \$propertyValue=null. The purpose of this process is to get the value of property defined by \$propertyName of the HTML element defined by \$htmlElement and return it in \$propertyValue. In case \$htmlElement is null (Figure 4.23, If-Then-Else-53), the process sets the focus in class HTMLObject (Perform-51) and asks from the author to select the HTML object whose property will be read (Perform-68). In our example, the author must select WebEq_Input_Control_1. Perform-138, Perform-148 and Perform-162 create the name of the property's instance in the MATHESIS ontology, WebEq-Input-Control-1_id in our example, and assign it to \$propertyName. Perform-183 actually reads the value of the *id* property, by getting the value of the data property html-property-value of instance WebEq-

Input-Control-1_id (see Figure 4.11). Execution control returns back to process define_code_to_initialize_interface_elements (Figure 4.22).

2. If-Then_Else-38(isNull(\$id)) checks whether the *id* property of the HTML element Web-Eq-Input-Control-1 actually has a value. In our example, this property has not been given a value by the author yet ($\$id=null$) and therefore needs to be assigned a value (Sequence-42). Perform-54 displays an input dialog prompting the author to enter the value of the *id* property of Web-Eq-Input-Control-1. The statement proposes as a default the name of the element itself. The author may enter “expressionInputControl” as the value of the *id* property for Web-Eq-Input-Control-1.
3. Perform-4(createInstance(\$id,mathesis:JavaScript-Variable-HTML-Reference)) creates the JavaScript variable that will reference Web-Eq-Input-Control_1, an instance of class JavaScript-Variable-HTML-Reference named expressionInputControl (Figure 4.21). Perform-44(setDataProperty(\$variable,hasScope,global)) sets the variable’s scope to global.
4. Perform-6(createInstance(any,JavaScript_Assignment)) creates a JavaScript-Assignment instance, JavaScript-Assignment-2 in our example, that will represent JavaScript statement `expressionInputControl = getElementById(“expressionInputControl”)`. The ontological representation of such an assignment statement is shown in Figure 4.24.

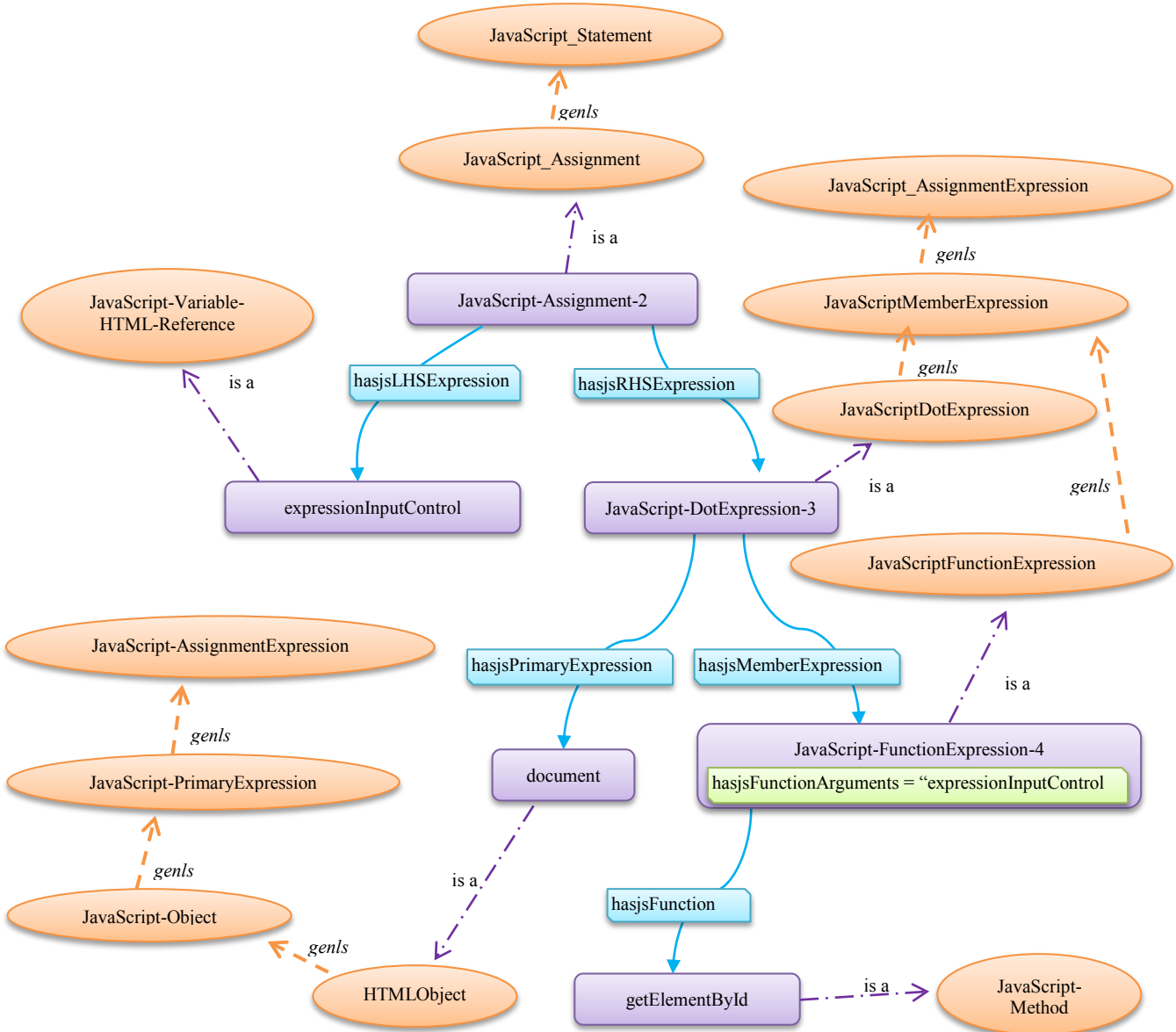


Fig. 4.24 Ontological representation of JavaScript statement `expressionInputControl=document.getElementById('expressionInputControl')`

5. Perform_90(setObjectProperty(\$assignment,hasjsLHSExpression,\$variable)) sets property hasjsLHSExpression of the JavaScript-Assignment-2 instance to point at the expressionInputControl instance of class JavaScript-Variable-HTML-Reference.
6. Perform-7(get_interface_element_reference(\$id,\$dotExpression) calls authoring process get_interface_element_reference (Figure 4.25) that creates an instance of class JavaScript_DotExpression, JavaScript-DotExpression-3 in Figure 4.24. This instance represents the right-hand side of the JavaScript statement, i.e. document.getElementById("expressionInputControl").
7. Perform-95 assigns instance JavaScript-DotExpression-3 as the right-hand side of instance JavaScript-Assignment-2. The ontological representation of JavaScript-Assignment-2 is now completed and the statement is ready to be added as the first statement of the tutoring process execute-monomial-multiplication-Presentation (Figure 4.1c, Perform_6). This is achieved by statement Perform_8(addPerform(\$assignment)).

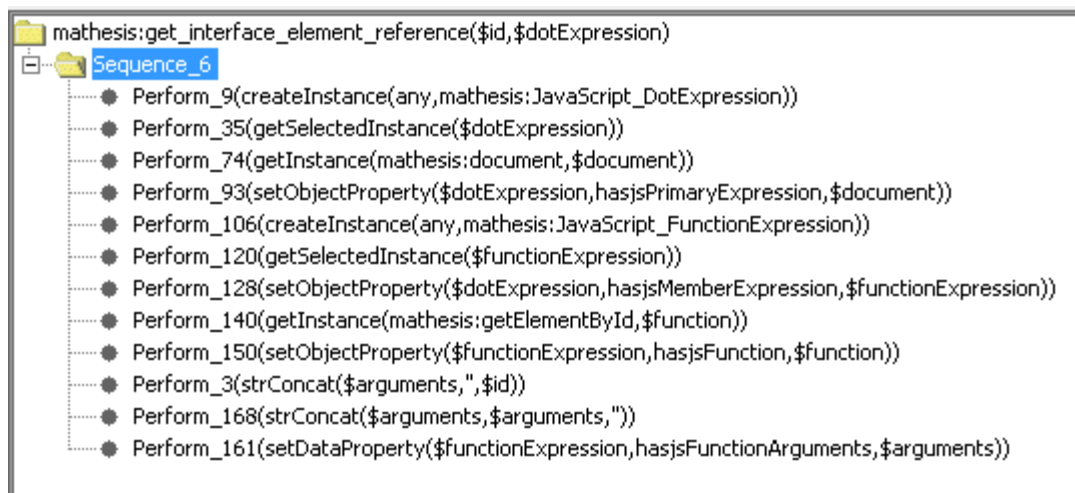


Fig. 4.25 Authoring process get_interface_element_reference

In the same way, JavaScript statement `answerInputControl = getElementById("answerInputControl")` is represented by instance `JavaScript-Assignment-3` and is added as the second statement of tutoring process `execute-monomial-multiplication-Presentation` as shown in Figure 4.26. At this point, the execution of authoring process `authoring-task-present-domain-task` (Figure 4.12) is completed.

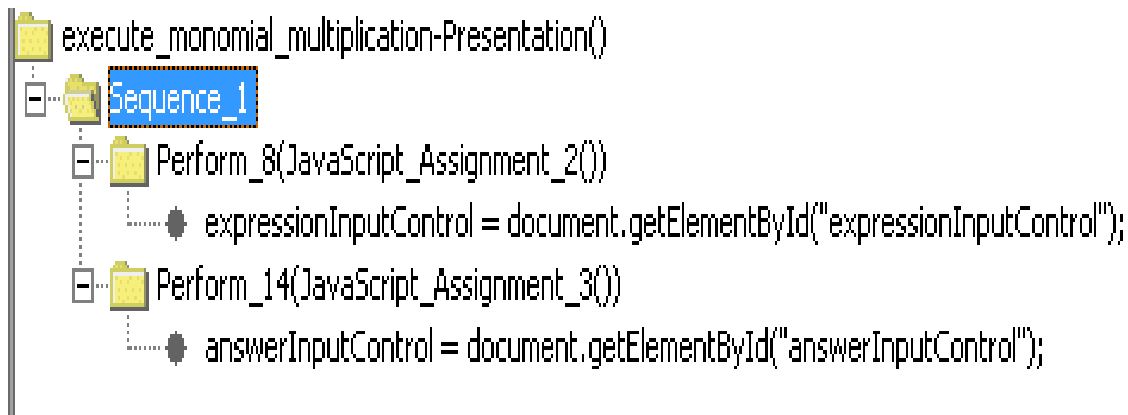


Fig. 4.26 Tutoring process `execute-monomial-multiplication-Presentation`

The detailed, step-by-step description of the execution of authoring processes as well as of the ontological representations they act on, clearly shows that it is technically possible to achieve the main research goal of the MATHESIS framework: to represent ontologically the whole knowledge that is needed to develop an Intelligent Tutoring System, declarative and procedural, domain and authoring, in order to perform reasoning on it, that is, making it inspect-able, reusable and reasonable.

Chapter 5

Chapter 5: Discussion

5.1 KNOWLEDGE REUSE AND SCALABILITY

Building from scratch any kind of knowledge-based system, like a model-tracing tutor, requires a tremendous effort, even with the use of authoring tools (Murray, 2003b). The main reason for this problem is the knowledge acquisition bottleneck. Domain experts are not trained in articulating their (teaching) expertise in a more abstract way. Highly trained authors are needed to perform the following knowledge engineering tasks: Develop ontologies, represent the curriculum, represent teaching strategies and diagnostic procedures and create student models.

Existing authoring tools fall in either side of the breadth vs. depth trade-off: a) Tools that build tutors for very specific domains but with a deep domain model, or b) all-purpose authoring shells that build tutors for various domains but with shallow domain knowledge models. In addition to this trade-off, the problem of knowledge reuse - both for the developed tutoring (domain) knowledge as well as for the authoring expertise - is

severely restraining the widespread use of model-tracing tutors in particular and knowledge based systems in general in the real world.

As a solution to these problems, a three tiered meta-authoring framework was proposed by Murray (2003b). At its base there is a generic ITS authoring tool that requires the top level of authoring expertise like knowledge/ontological engineering, cognitive task analysis, instructional design, learning theories. This system is used to develop the middle tier, that is, reusable libraries of domain ontologies, student modeling rules, interface templates and generic teaching strategies. At the top level there are fairly simple tools for trained teachers that make use of the reusable libraries to develop powerful tutors for real world use. It is this meta-authoring model that the MATHESIS framework implements:

- a) The base level is the ONTOMATH language and the Meta-Authoring tools used by meta-authors to represent authoring expertise as an ontology of executable authoring processes. These processes can guide domain experts in performing any kind of authoring tasks.
- b) The middle tier corresponds to the MATHESIS Ontology, consisting of reusable parts of the tutors' cognitive, teaching and interface models. In the case of the monomial multiplication tutor development presented in section 4, these reusable parts where generic instances like monomial (domain concept), the ModelTracingAlgorithm (tutoring model), WebEq-Input-Control (HTML element) as well as reusable authoring processes like the assign_js_method (JavaScript programming process). These generic instances are used by either the authoring tools or the authoring processes to create specific instances that represent the various parts of the monomial multiplication tutor.

- c) The top level comprises the domain-specific authoring tools, model-tracing in our case. The key characteristic of these tools is that they facilitate trained teachers (domain experts) to browse, locate and (re-)use the components of the middle tier. In the case of the MATHESIS model-tracing tutor authoring tools (Figure 4.1), this facilitation depends on the depth of the authoring ONTOMATH model, as it is the authoring processes that guide domain experts in locating and reusing existing or creating new tutors' parts. In that sense, each authoring process is an authoring tool at a different level. This multi-level classification of authoring processes as authoring tools is illustrated even better by the fact that each one of the tools for tutor creation/selection, domain task creation/selection and teaching strategy model selection are shortcuts to common, top level authoring actions. They could be easily removed from the interface and replaced by authoring processes that would perform the same authoring actions. The same holds for middle or low level authoring processes (tools). The more fine-grained they become the more authoring expertise they give to the system and the less expertise they demand from the domain expert.

The problem of the authoring model's granularity was faced in an evaluation of the system. A trained teacher of mathematics and expert computer user was asked to use the tools to re-implement the monomial multiplication task of the MATHESIS Algebra Tutor. After 35 hours of training with the tools, he was capable of executing the authoring model and re-implement the monomial multiplication tutor. Then he tried to implement a monomial division tutor by executing the same model. Monomial division was selected as it has minor differences from the multiplication task, mainly in the domain expertise model. When the author tried to implement tutoring process

DivideMainParts that would perform the monomial division, the counterpart of tutoring process MultiplyMainParts (Figure 4.9), he had to modify the MultiplyMainParts process so that it would subtract the exponents of common variables ($x^m \div x^n = x^{m-n}$) rather than adding them ($x^m \cdot x^n = x^{m+n}$). In the MATHESIS framework there are two ways to solve this problem:

- i. The non-expert author must use the Advanced Authoring Tools for Tutoring Domain Processes (Figure 3.7) to change the representation of the JavaScript statement that adds the exponents, so as to perform a subtraction instead of an addition. However, this elementary change entails a considerable raise at the expertise level required by the author; he must identify that statement (knowledge of JavaScript and programming) and change directly its ontological representation (knowledge of the MATHESIS ontology).
- ii. The meta-author develops authoring processes that guide the domain expert in defining mathematical operations between mathematical objects. Then, the domain expert is guided by these processes to define whether the exponents (integers) of common variables are added (monomial multiplication), subtracted (monomial division) or even multiplied (monomial powers), like in $(x^m)^n = x^{m \cdot n}$.

Therefore, the MATHESIS framework deepens Murray's three tier structure of the tools to an arbitrary depth. Any time that the MATHESIS framework must cover the creation of tutors in a new domain, meta-authors must create both the generic instances that represent declarative and procedural knowledge of the new domain as well as the authoring processes that use them. Of course, parts of the new tutor(s) that are common with already developed tutors are readily reusable. In the case of the monomial division

tutor monomial, ModelTracingAlgorithm and WebEq-Input-Control can be reused along with the authoring processes that use them. This is possible due to the representation of the authoring processes via the ONTOMATH language ontologically, that is in a declarative form, which makes them open for inspection and therefore mostly reusable (Gómez-Pérez, Fernández-López, Corcho, 2004). In existing authoring systems, adding new authoring knowledge would entail modifying the authoring program and adding new tools with their corresponding interfaces. This modification can be done only by the creators of the authoring program.

5.2 CONCLUSIONS AND FURTHER WORK

The MATHESIS meta-authoring engineering framework primary goal is to spread the load of authoring over various levels of reusable authoring processes and over various authors that can reuse them by browsing, locating and modifying them. For example, the ontological representation of any programming language like HTML or JavaScript and the authoring processes that create these representations can be standardized by authorised organizations and then used by any meta-author. Experts from any domain can develop libraries of ontological representations for their tutors and the authoring processes that create them. They could even develop their customized meta-authoring and domain-specific authoring tools and distribute them as Java applets. The implementation of the MATHESIS framework as an all-in-one package inside the Protégé OWL editor was done for easing the implementation of the system. The framework could have been implemented with its parts distributed: The MATHESIS ontology being still an OWL ontology and the authoring tools being Java applications. Even Protégé's API used for grounding the ONTOMATH statements is a Java library. This distributed and collaborative

authoring scheme can be supported by modern tools such as the WebProtégé collaborative ontology editor and knowledge acquisition tool (Tudorache, Nyulas, Noy, & Musen, 2013).

Of course, this multi-layered meta-authoring framework doesn't force the knowledge acquisition bottleneck and the efforts of knowledge engineering to disappear. The ontological representation of the tutor's (KBS) models, the structure of the authoring processes and the execution details that have to be taken into consideration for their development, suggest that there is no royal road to knowledge engineering. For the moment, the MATHESIS ontology contains the authoring knowledge just for the development of two model tracing tutors, one for multiplication and one for division of monomials. Therefore, it is not claimed that the problem of knowledge acquisition and reuse was solved. Being in an experimental stage, the MATHESIS framework must be considered as a *proof-of-concept* system. To investigate the issues of knowledge reuse and scalability, authoring models for monomial by polynomial multiplication and polynomial multiplication must be developed. In the long term, an authoring model that could create the entire MATHESIS Algebra Tutor should be developed. To facilitate this procedure, new authoring tools are needed, such as parsers for transforming between HTML and MATHESIS DOM representation; parsers for transforming between JavaScript and MATHESIS tutoring processes; extension of the ONTOMATH language and elaboration of its interpreter. These tools call for further research.

Appendix A

Appendix A: Complete Math Domain Cognitive Model of the MATHESIS Algebra Tutor

A1. MONOMIAL MULTIPLICATION:

1. Multiply monomials: $3x^2y \cdot (-4xz^3) = -12x^3yz^3$

1.1. Multiply coefficients: $3 \cdot (-4) = -12$

1.2. Multiply main parts:

1.2.1. Add exponents of common variables: $x^2 \cdot x = x^{2+1} = x^3$

1.2.2. Copy exponents of single variables: $y \cdot z^3 = yz^3$

1.2.3. Multiply all variables (common error): $3x^2y \cdot (-4xz^3) = -12x^3z^3$

1.2.4. Add one for common variables not having exponents (common error)

: $3x^2y \cdot (-4xz^3) = -12x^2yz^3$

1.2.5. Do not introduce non-existent variables (common error): $3x^2y \cdot$

$(-4xz^3) = -12x^3yz^3$

A2. MONOMIAL DIVISION

2. Divide monomials: $\frac{-12x^3yz^3}{3x^2y} = -4xz^3$

2.1. Divide coefficients: $-12:3 = -4$

2.2. Divide main parts:

2.2.1. Subtract exponents of common variables: $x^3:x^2 = x^{3-2} = x$ and $y:y = y^{1-1} = y^0 = 1$

2.2.2. Copy exponents of single variables: $z^3 = z^3$

2.2.3. Divide all variables (common error): $\frac{-12x^3yz^3}{3x^2y} = -4z^3$

2.2.4. Subtract one for common variables not having exponents (common

error): $\frac{-12x^3yz^3}{3x^2y} = -4xy$

- 2.2.5. Do not introduce non-existent variables (common error):

$$\frac{-12x^3yz^3}{3x^2y} = -4xwz^3$$

- 2.2.6. Eliminate variables with equal exponents (zero exponent) (common error):

$$\frac{-12x^3yz^3}{3x^2y} = -4xy$$

A3. COLLECTION OF LIKE TERMS

3. Collect like terms:

- 3.1. Recognize two identical monomials (“like terms”)

3.1.1. Every variable of the first one must be present in the second one

3.1.2. Every variable of the second one must be present in the first one

3.1.3. Common variables must have the same exponents

- 3.2. Group Identical Monomials by marking them (same group number):

$$\underbrace{2xy}_1 \underbrace{-x^2}_2 + \underbrace{7xy}_1 \underbrace{+6x^2}_2$$

- 3.2.1. All monomials in a group must be identical to the first term of the group (3.1)

- 3.3. Add the coefficients of each group: $2 + 7 = 9$ and $-1 + 6 = 5$

- 3.4. Keep the main part of each group: $2xy + 7xy = 9xy$ and $-x^2 + 6x^2 = 5x^2$

A4. MONOMIAL POWER

4. Raise monomial to power: $(-2x^2yz^3)^3 = -8x^6y^3z^9$

- 4.1. Raise the coefficient to the power: $(-2)^3 = -8$

- 4.2. Raise main part to the exponent:

4.2.1. Multiply the exponents: $(x^2yz^3)^3 = x^{2 \cdot 3}y^{1 \cdot 3}z^{3 \cdot 3} = x^6y^3z^9$

4.2.2. Raise all variables (common error): $(-2x^2yz^3)^3 = -8x^6z^9$

- 4.2.3. Do not introduce non-existent variables (common error).

A5. MONOMIAL BY POLYNOMIAL MULTIPLICATION

5. Multiply monomial by polynomial: $3x^3y \cdot (2x - y^2z) = 6x^4y - 3x^3y^3z$

5.1. Identify the monomial terms of the polynomial: $2x$ and $-y^2z$

5.2. Multiply each one of them with the monomial (A1):

5.3. $3x^3y \cdot 2x = 6x^4y$ and $3x^3y \cdot (-y^2z) = -3x^3y^3z$

NOTICE: In the current implementation the tutor displays automatically the partial products that the student must execute and therefore there can be no assessment of the application of the distributive property.

A6. POLYNOMIAL BY POLYNOMIAL MULTIPLICATION

6. Multiply polynomial by polynomial:

$$(3xy - 2x^2) \cdot (2x^2y^2 - 4xy) = 6x^3y^3 - 12x^2y^2 - 4x^4y^2 + 8x^3y$$

6.1. Identify the monomial terms of the first polynomial: $3xy$ and $-2x^2$

6.2. Identify the monomial terms of the second polynomial: $2x^2y^2$ and $-4xy$

6.3. Multiply each term of the first monomial with each term of the second monomial

$$(A1): 3xy \cdot 2x^2y^2 = 6x^3y^3 \text{ and } 3xy \cdot (-4xy) = -12x^2y^2 \text{ and } -2x^2 \cdot$$

$$2x^2y^2 = -4x^4y^2 \text{ and } -2x^2 \cdot (-4xy) = 8x^3y$$

NOTICE: In the current implementation the tutor displays automatically the partial products that the student must execute and therefore there can be no assessment of the application of the distributive property.

A7 PARENTHESES' ELIMINATION

7. Eliminate parentheses:

$$(5x^2 - 8xy + 3) - (x^2 + 4xy - 5) =$$

$$5x^2 - 8xy + 3 - x^2 - 4xy + 5 =$$

$$4x^2 - 12xy + 8$$

7.1. Keep the sign of each parenthesized term if the sign in front of the parenthesis is

a plus (+): $(5x^2 - 8xy + 3) = 5x^2 - 8xy + 3$

7.2. Change the sign of each parenthesized term if the sign in front of the parenthesis

is a minus (-): $-(x^2 + 4xy - 5) = -x^2 - 4xy + 5$

7.3. Collect like terms if there are any (A3): $5x^2 - 8xy + 3 - x^2 - 4xy + 5 =$

$$4x^2 - 12xy + 8$$

A8. SQUARE OF SUM/DIFFERENCE EXPANSION

8. Expand Square of Sum/Difference: $(2x \pm 3)^2 = 4x^2 \pm 12x + 9$

8.1. Recall the expanded form of the identity: $(a \pm b)^2 = a^2 \pm 2ab + b^2$

NOTICE: Because of the complexity of this task, the tutor displays a visual scaffold of the expanded form of the identity: $\square^{\square} \pm 2 \cdot \square \cdot \square + \square^{\square}$

8.2. Substitute a and b for the real terms (a = 2x and b = 3): $(2x + 3)^2 = (2x)^2 + 2 \cdot 2x \cdot 3 + 3^2$

8.2.1. Write the first term of the sum/difference, squared: $\boxed{(2x)}^{\boxed{2}} \pm 2 \cdot \boxed{} \cdot \boxed{} + \boxed{}^{\boxed{}}$

8.2.2. Write the first term of the sum/difference as the second term of the double product: $\boxed{(2x)}^{\boxed{2}} \pm 2 \cdot \boxed{2x} \cdot \boxed{} + \boxed{}^{\boxed{}}$

8.2.3. Write the second term of the sum/difference as the third term of the double product: $\boxed{(2x)}^{\boxed{2}} \pm 2 \cdot \boxed{2x} \cdot \boxed{3} + \boxed{}^{\boxed{}}$

8.2.4. Write the second term of the sum/difference, squared: $\boxed{(2x)}^2 \pm 2 \cdot \boxed{2x} \cdot \boxed{3} + \boxed{3}^2$

8.2.5. Take care for parenthesized terms (common error): $(2x + 3)^2 = 2x^2 + 2 \cdot 2x \cdot 3 + 3^2$

8.3. Perform monomial multiplications and powers (A1 and A4): $(2x)^2 + 2 \cdot 2x \cdot 3 + 3^2 = 4x^2 + 12x + 9$

A9. PRODUCT OF SUM BY DIFFERENCE EXPANSION

9. Expand Product of Sum by Difference: $(2x + 3) \cdot (2x - 3) = 4x^2 - 9$

9.1. Recall the expanded form of the identity: $(a + b) \cdot (a - b) = a^2 - b^2$

NOTICE: Because of the complexity of this task, the tutor displays a visual scaffold of the expanded form of the identity: $\boxed{}^{\boxed{}} - \boxed{}^{\boxed{}}$

9.2. Substitute a and b for the real terms (a = 2x and b = 3): $(2x + 3) \cdot (2x - 3) = (2x)^2 - 3^2$

9.2.1. Write the first term of the sum/difference, squared: $\boxed{(2x)}^2 - \boxed{}^{\boxed{}}$

9.2.2. Write the second term of the sum/difference, squared: $\boxed{(2x)}^2 - \boxed{3}^2$

9.2.3. Take care for parenthesized terms (common error): $(2x + 3) \cdot (2x - 3) = 2x^2 - 9$

9.3. Perform monomial powers (A4): $(2x)^2 - 3^2 = 4x^2 - 9$

A10. CUBE OF SUM/DIFFERENCE EXPANSION

10. Expand cube of sum/difference: $(2x - 3)^3 = 8x^3 - 36x^2 + 54x - 27$

10.1. Recall the expanded form of the identity: $(a \pm b)^3 = a^3 \pm 3 \cdot a^2 \cdot b + 3 \cdot a \cdot b^2 \pm b^3$

NOTICE: Because of the complexity of this task, the tutor displays a visual scaffold of the expanded form of the identity: $\square^3 \pm 3 \cdot \square^2 \cdot \square + 3 \cdot \square \cdot \square^2 \pm \square^3$

10.2. Substitute a and b for the real terms (a = 2x and b = 3): $(2x - 3)^3 = (2x)^3 - 3 \cdot (2x)^2 \cdot 3 + 3 \cdot 2x \cdot 3^2 - 3^3$

10.2.1. Write the first term of the sum/difference, cubed: $\boxed{(2x)}^3 \pm 3 \cdot \boxed{}^2 \cdot \boxed{} + 3 \cdot \boxed{} \cdot \boxed{}^2 \pm \boxed{}^3$

10.2.2. Write the first term of the sum/difference squared, as the second term of the first triple product: $\boxed{(2x)}^3 \pm 3 \cdot \boxed{(2x)}^2 \cdot \boxed{} + 3 \cdot \boxed{} \cdot \boxed{}^2 \pm \boxed{}^3$

10.2.3. Write the second term of the sum/difference as the third term of the first triple product: $\boxed{(2x)}^3 \pm 3 \cdot \boxed{(2x)}^2 \cdot \boxed{3} + 3 \cdot \boxed{} \cdot \boxed{}^2 \pm \boxed{}^3$

10.2.4. Write the first term of the sum/difference as the second term of the second triple product: $\boxed{(2x)}^3 \pm 3 \cdot \boxed{(2x)}^2 \cdot \boxed{3} + 3 \cdot \boxed{2x} \cdot \boxed{}^2 \pm \boxed{}^3$

10.2.5. Write the second term of the sum/difference squared, as the third term of the second triple product: $\boxed{(2x)}^3 \pm 3 \cdot \boxed{(2x)}^2 \cdot \boxed{3} + 3 \cdot \boxed{2x} \cdot \boxed{3}^2 \pm \boxed{}^3$

10.2.6. Write the second term of the sum/difference, cubed: $\boxed{(2x)}^3 \pm 3 \cdot \boxed{(2x)}^2 \cdot \boxed{3} + 3 \cdot \boxed{2x} \cdot \boxed{3}^2 \pm \boxed{3}^3$

10.2.7. Take care for parenthesized terms (common error): $(2x - 3)^3 = 2x^3 \pm 3 \cdot 2x^2 \cdot 3 + 3 \cdot 2x \cdot 3^2 \pm 3^3$

10.2.8. Perform monomial multiplications and powers (A1 and A4): $(2x)^3 - 3 \cdot (2x)^2 \cdot 3 + 3 \cdot 2x \cdot 3^2 - 3^3 = 8x^3 - 36x^2 + 54x - 27$

A11. FACTORING BY COMMON FACTOR

11. *Factoring by common factor:* $2x^3(x + 1) - 4x^2(x + 1)^2 + 6x^2y(x + 1) = 2x^2(x - 2 + 3y)$

11.1. Find the common factor: $2x^2(x + 1)$

11.1.1. Find the GCD of the coefficients: $GCD(2,4,6)=2$

11.1.2. Find the GCD of common variables: $GCD(x^3, x^2, x^2) = x^2$

11.1.3. Find the DCG of common parentheses: $GCD((x + 1), (x + 1)^2, (x + 1)) = (x + 1)$

11.2. Divide terms by the common factor:

$$\frac{2x^3(x + 1)}{2x^2(x + 1)} = x \text{ and } \frac{-4x^2(x + 1)^2}{2x^2(x + 1)} = -2(x + 1) \text{ and } \frac{6x^2y(x + 1)}{2x^2(x + 1)} = 3y$$

11.2.1. Divide numerical coefficients

11.2.2. Subtract exponents of common variables.

11.2.3. Subtract exponents of common parentheses

NOTICE: Due to the complexity of the cognitive tasks described above, the tutor automatically displays templates for both the common factor and the partial quotients, indicating the kind of terms that appear in these results: numbers, variables with exponents, parentheses with exponents. In addition, the tutor automatically writes progressively the product of the common factor by the parenthesis containing the partial quotients. The template for the common factor $2x^2(x + 1)$ is $\square \cdot \square^{\square} \cdot (\square)^{\square}$ while the template for partial quotient $\frac{-4x^2(x+1)^2}{2x^2(x+1)} = -2(x + 1)$ is $\square \cdot (\square)^{\square}$

A12 FACTORING BY DIFFERENCE OF SQUARES

12. Factor using Difference of Squares identity: $4x^2 - 9 = (2x)^2 - 3^2 = (2x + 3) \cdot (2x - 3)$

12.1. Find the square roots of the two terms: $\sqrt{4x^2} = 2x$ and $\sqrt{9} = 3$

12.1.1. Find the square root of the numerical coefficient: $\sqrt{4} = 2$ and $\sqrt{9} = 3$

12.1.2. Check that the exponents of all variables are even and divide them by two: $x^2 = (x^1)^2$

12.1.3. Check that the exponents of all parenthesized factors are even and divide them by two

12.1.4. NOTICE: The factored form of the identity is given automatically as a template $(\square + \square) \cdot (\square - \square)$

12.2. Substitute the square roots into the template:

12.2.1. Enter the square root of the first term as the first term of the sum:
 $(\boxed{2x} + \square) \cdot (\square - \square)$

12.2.2. Enter the square root of the second term as the second term of the sum:
 $(\boxed{2x} + \boxed{3}) \cdot (\square - \square)$

12.2.3. Enter the square root of the first term as the first term of the difference:
 $(\boxed{2x} + \boxed{3}) \cdot (\boxed{2x} - \square)$

12.2.4. Enter the square root of the second term as the second term of the difference:
 $(\boxed{2x} + \boxed{3}) \cdot (\boxed{2x} - \boxed{3})$

A13. FACTORING BY SUM OF CUBES

13. Factor using Sum of Cubes identity: $8x^3 + 27 = (2x)^3 + 3^3 = (2x + 3)(4x^2 - 6x + 9)$

- 13.1. Find the cubic roots of the two terms: $\sqrt[3]{8x^3} = 2x$ and $\sqrt[3]{27} = 3$
- 13.1.1. Find the cubic root of the numerical coefficient: $\sqrt[3]{8} = 2$ and $\sqrt[3]{27} = 3$
- 13.1.2. Check that the exponents of all variables are multiples of three and divide them by 3: $x^3 = (x^1)^3$
- 13.1.3. Check that the exponents of all parenthesised factors are multiples of three and divide them by 3

NOTICE: The factored form of the identity is given automatically as a template

$$(\square + \square) \cdot (\square^2 - 2 \cdot \square \cdot \square + \square^2)$$

- 13.2. Substitute the cubic roots into the template:
- 13.2.1. Enter the cubic root of the first term as the first term of the first parenthesis: $(\boxed{2x} + \square) \cdot (\square^2 - 2 \cdot \square \cdot \square + \square^2)$
- 13.2.2. Enter the cubic root of the second term as the second term of the first parenthesis: $(\boxed{2x} + \boxed{3}) \cdot (\square^2 - 2 \cdot \square \cdot \square + \square^2)$
- 13.2.3. Enter the cubic root of the first term squared, as the first term of the second parenthesis: $(\boxed{2x} + \boxed{3}) \cdot (\boxed{(2x)}^2 - 2 \cdot \square \cdot \square + \square^2)$
- 13.2.4. Enter the cubic root of the first term as the first term of the product: $(\boxed{2x} + \boxed{3}) \cdot (\boxed{(2x)}^2 - 2 \cdot \boxed{2x} \cdot \square + \square^2)$
- 13.2.5. Enter the cubic root of the second term as the second term of the product: $(\boxed{2x} + \boxed{3}) \cdot (\boxed{(2x)}^2 - 2 \cdot \boxed{2x} \cdot \boxed{3} + \square^2)$
- 13.2.6. Enter the cubic root of the second term squared, as the last term of the second parenthesis: $(\boxed{2x} + \boxed{3}) \cdot (\boxed{(2x)}^2 - 2 \cdot \boxed{2x} \cdot \boxed{3} + \boxed{3}^2)$
- 13.3. Perform monomial multiplications and powers (A1 and A4): $(2x + 3) \cdot ((2x)^2 - 2 \cdot 2x \cdot 3 + 3^2) = (2x + 3)(4x^2 - 6x + 9)$

A14. FACTORING BY DIFFERENCE OF CUBES

14. Factor using Difference of Cubes identity: $8x^3 - 27 = (2x)^3 - 3^3 = (2x - 3)(4x^2 + 6x + 9)$

14.1. Find the cubic roots of the two terms: $\sqrt[3]{8x^3} = 2x$ and $\sqrt[3]{27} = 3$

14.1.1. Find the cubic root of the numerical coefficient: $\sqrt[3]{8} = 2$ and $\sqrt[3]{27} = 3$

14.1.2. Check that the exponents of all variables are multiples of three and divide them by 3: $x^3 = (x^1)^3$

14.1.3. Check that the exponents of all parenthesised factors are multiples of three and divide them by 3

NOTICE: The factored form of the identity is given automatically as a template $(\square - \square) \cdot (\square^2 + 2 \cdot \square \cdot \square + \square^2)$

14.2. Substitute the cubic roots into the template:

14.2.1. Enter the cubic root of the first term as the first term of the difference:

$$(\boxed{2x} - \square) \cdot (\square^2 + 2 \cdot \square \cdot \square + \square^2)$$

14.2.2. Enter the cubic root of the second term as the second term of the difference: $(\boxed{2x} - \boxed{3}) \cdot (\square^2 + 2 \cdot \square \cdot \square + \square^2)$

14.2.3. Enter the cubic root of the first term squared, as the first term of the second parenthesis: $(\boxed{2x} - \boxed{3}) \cdot (\boxed{(2x)}^2 + 2 \cdot \square \cdot \square + \square^2)$

14.2.4. Enter the cubic root of the first term as the first term of the product:

$$(\boxed{2x} - \boxed{3}) \cdot (\boxed{(2x)}^2 + 2 \cdot \boxed{2x} \cdot \square + \square^2)$$

14.2.5. Enter the cubic root of the second term as the second term of the product: $(\boxed{2x} - \boxed{3}) \cdot (\boxed{(2x)}^2 + 2 \cdot \boxed{2x} \cdot \boxed{3} + \square^2)$

14.2.6. Enter the cubic root of the second term squared, as the last term of the second parenthesis: $(\boxed{2x} - \boxed{3}) \cdot (\boxed{(2x)}^2 - 2 \cdot \boxed{2x} \cdot \boxed{3} + \boxed{3}^2)$

- 14.3. Perform monomial multiplications and powers (A1 and A4): $(2x - 3) \cdot ((2x)^2 + 2 \cdot 2x \cdot 3 + 3^2) = (2x - 3)(4x^2 + 6x + 9)$

A15. FACTORING BY SQUARE OF SUM/DIFFERENCE

15. Factor an expanded Square of Sum/Difference: $4x^2 \pm 24x + 9 = (2x)^2 \pm 2 \cdot 2x \cdot 3 + 3^2 = (2x \pm 3)^2$

- 15.1. Find the square roots of the first and third term (A12.3.1): $\sqrt{4x^2} = 2x$ and $\sqrt{9} = 3$

- 15.2. NOTICE: The factored form of the identity is given automatically as a template $(\square \pm \square)^2$.

- 15.3. Verify that the middle term is the double product of the square roots of the first and third term: $24x = 2 \cdot 2x \cdot 3$

- 15.4. Substitute the square roots of the terms into the template:

- 15.4.1. Enter the square root of the first term as the first term of the sum/difference: $(\boxed{2x} \pm \square)^2$.

- 15.4.2. Enter the square root of the second term as the second term of the sum/difference: $(\boxed{2x} \pm \boxed{3})^2$.

A16. FACTORING THE QUADRATIC FORM

16. Factor the quadratic form:

$$x^2 + Sx + P = (x + a)(x + b), a \cdot b = P \text{ and } a + b = S:$$

$$x^2 + 5x + 6 = (x + 2)(x + 3)$$

- 16.1. Identify $P = a \cdot b$ and $S = a + b$: $P = 6$ and $S = 5$

- 16.2. Find the pairs of integers a, b that have a product of $P = 6$:

16.3. $1 \cdot 6 = 6$ or $(-1) \cdot (-6) = 6$ or $2 \cdot 3 = 6$ or $(-2) \cdot (-3) = 6$

16.4. Find the pair a, b that gives a sum of $S = 5$: $2 + 3 = 5$

16.5. Write the factored form: $(x + a)(x + b) = (x + 2)(x + 3)$

A17. FACTORING BY TERM GROUPING

NOTICE: Grouping is the most difficult of the factoring techniques as it demands backtracking, according to the next algorithm:

$$\begin{aligned} x^4 - 1 + x^3 - x &= (x^2 - 1)(x^2 + 1) + x(x^2 - 1) \\ &= (x^2 - 1)[(x^2 + 1) + x] = (x - 1)(x + 1)(x^2 + x + 1) \end{aligned}$$

17.1 Group the terms in various groups. The number of terms in each group depends on the total number of terms: $x^4 - 1$ and $x^3 - x$

17.2 Factor each group separately using any of the previous factoring methods:

$$\begin{aligned} x^4 - 1 &= (x^2 - 1)(x^2 + 1) && \text{(Difference of squares)} \\ x^3 - x &= x(x^2 - 1) && \text{(CommonFactor)} \end{aligned}$$

17.3 Check that the products that come from each group now have a common factor (always a parenthesis). If not, BACKTRACK to step 17.1 and change the groups: $(x^2 - 1)$ is the common factor of the two groups.

17.4 If there is a common factor, use the common factor method to factor the expression: $(x^2 - 1)(x^2 + 1) + x(x^2 - 1) = (x^2 - 1)[(x^2 + 1) + x] = (x - 1)(x + 1)(x^2 + x + 1)$

The tutor supports all these steps separately but it doesn't provide automatic checking for a common factor and backtracking from 17.3 to 17.1. The student must manually restart the grouping and re-group the terms.

Appendix B

Appendix B: The ONTOMATH Atomic Authoring Statements Reference

B1. ONTOMATH_BROWSE STATEMENTS

Statement	Purpose	Parameters		
		Name	Type	I/O
<i>setSelectedClass</i> (<i>classNameIdentifier</i>)	Sets the Class named by <i>classNameIdentifier</i> as selected in the Classes Panel	<i>classNameIdentifier</i>	String Literal/Variable	Input
<i>getSelectedClass</i> (<i>classNameIdentifier</i>)	Sets <i>classNameIdentifier</i> to the name of the Class selected in the Classes Panel	<i>classNameIdentifier</i>	String Variable	Output
<i>setSelectedInstance</i> (<i>instanceNameIdentifier</i>)	Sets the Instance named by <i>instanceNameIdentifier</i> as selected in the Instances Panel	<i>instanceNameIdentifier</i>	String Literal/Variable	Input
<i>getSelectedInstance</i> (<i>instanceNameIdentifier</i>)	Sets <i>instanceNameIdentifier</i> to the name of the Instance selected in the Instances Panel	<i>instanceNameIdentifier</i>	String Variable	Output
<i>getInstance</i> (<i>instanceNameIdentifier</i> , <i>instanceReference</i>)	Sets <i>instanceReference</i> to point to the instance named by <i>instanceNameIdentifier</i>	<i>instanceNameIdentifier</i>	String Literal/Variable	Input
		<i>instanceReference</i>	DefaultOWLIndividual Variable	Output

B2. ONTOMATH_COLLECTION STATEMENTS

Statement	Purpose	Parameters		
		Name	Type	I/O
<i>iteratorNext</i> (<i>iteratorIdentifier</i> , <i>elementIdentifier</i>)	Sets <i>elementIdentifier</i> to the next element of <i>iteratorIdentifier</i>	<i>iteratorIdentifier</i>	Iterator Variable	Input
		<i>elementIdentifier</i>	OWLIndividual Variable	Output

B3. ONTOMATH_STRING STATEMENTS

Statement	Purpose	Parameters		
		Name	Type	I/O
<i>strConcat</i> (<i>newString, string1, string2</i>)	Concatenates <i>string1</i> and <i>string2</i> and returns the concatenated string to <i>newString</i>	<i>newString</i>	String Variable	Output
		<i>string1</i>	String Literal/Variable	Input
		<i>string2</i>	String Literal/Variable	Input

B4. ONTOMATH_DIALOG STATEMENTS

Statement	Purpose	Parameters		
		Name	Type	I/O
<i>showMessageDialog</i> (<i>message</i>)	Displays a Message Dialog with <i>message</i>	<i>message</i>	String Literal/Variable	Input
<i>showInputDialog</i> (<i>userInput, message, defaultValue</i>)	Displays an Input Dialog with <i>message</i> and <i>defaultValue</i> . Returns user input to <i>userInput</i>	<i>userInput</i>	String Variable	Output
		<i>message</i>	String Literal/Variable	Input
		<i>defaultValue</i>	String Literal/Variable	Input

B5. ONTOMATH_ONTOLOGY_EDITING STATEMENTS

Statement	Purpose	Parameters		
		Name	Type	I/O
<i>createInstance</i> (<i>instanceName</i> , <i>className</i>)	Creates a new instance of class <i>className</i> named <i>instanceName</i> . After creation the instance is set as selected in the Instances Panel. If <i>instanceName</i> is "null" , the author is asked for the name of the instance. If <i>instanceName</i> is "any" , a random name is given by the Protege system.	instanceName	String Literal/Variable	Input
		className	String Literal/Variable	Input
<i>copySelectedInstance</i> (<i>newInstanceName</i>)	Creates a new copy of the instance that is currently selected in the Instances Panel with name <i>newInstanceName</i> . If <i>newInstanceName</i> is "null" , the author is asked for the name of the new, copied instance.	newInstanceName	String Literal/Variable	Input
<i>createSubclass</i> (<i>subclassName</i> , <i>superclassName</i>)	Creates a new subclass named <i>subclassName</i> of class <i>superclassName</i> . After creation, the subclass is set as selected in the Classes Panel. If <i>subclassName</i> is "null" , the author is asked for the name of the subclass.	subclassName	String Literal/Variable	Input
		superclassName	String Literal/Variable	Input
<i>getObjectProperty</i> (<i>instanceIdentifier</i> , <i>propertyIdentifier</i> , <i>propertyValues</i>)	Gets the values of object property <i>propertyIdentifier</i> of instance <i>instanceIdentifier</i> and stores them to variable <i>propertyValues</i> . If the property is functional (it has only one value) then <i>propertyValues</i> is a single DefaultOWLIndividual. Otherwise, <i>propertyValues</i> is an Iterator containing the property's values.	instanceIdentifier	String Literal Or DefaultOWLIndividual Variable	Input
		propertyIdentifier	String Literal	Input
		propertyValues	DefaultOWLIndividual Or Iterator Variable	Output

Statement	Purpose	Parameters		
		Name	Type	I/O
<i>setObjectProperty</i> (<i>instanceIdentifier</i> , <i>propertyIdentifier</i> , <i>valueIdentifier</i>)	If object property <i>propertyIdentifier</i> of instance <i>instanceIdentifier</i> is functional (takes only one value), its value is set to <i>valueIdentifier</i> . Otherwise, <i>valueIdentifier</i> is added to the list of the property's values.	<i>instanceIdentifier</i>	String Literal Or DefaultOWLIndividual Variable	Input
		<i>propertyIdentifier</i>	String Literal	Input
		<i>valueIdentifier</i>	String Literal Or DefaultOWLIndividual Variable	Input
<i>getDataProperty</i> (<i>instanceIdentifier</i> , <i>propertyIdentifier</i> , <i>propertyValues</i>)	Gets the values of data property <i>propertyIdentifier</i> of instance <i>instanceIdentifier</i> and stores them to variable <i>propertyValues</i> . If the property is functional (it has only one value) then <i>propertyValues</i> is a single DefaultOWLIndividual. Otherwise, <i>propertyValues</i> is an Iterator containing the property's values.	<i>instanceIdentifier</i>	String Literal Or DefaultOWLIndividual Variable	Input
		<i>propertyIdentifier</i>	String Literal	Input
		<i>propertyValues</i>	DefaultOWLIndividual Or Iterator Variable	Output
<i>setDataProperty</i> (<i>instanceIdentifier</i> , <i>propertyIdentifier</i> , <i>valueIdentifier</i>)	If data property <i>propertyIdentifier</i> of instance <i>instanceIdentifier</i> is functional (takes only one value), its value is set to <i>valueIdentifier</i> . Otherwise, <i>valueIdentifier</i> is added to the list of the property's values. If <i>valueIdentifier</i> is "null" the author is asked for the new value.	<i>instanceIdentifier</i>	String Literal Or DefaultOWLIndividual Variable	Input
		<i>propertyIdentifier</i>	String Literal	Input
		<i>valueIdentifier</i>	String Literal/Variable	Input

Statement	Purpose	Parameters		
		Name	Type	I/O
<i>removePropertyValue</i> (<i>instanceIdentifier</i> , <i>propertyIdentifier</i> , <i>valueIdentifier</i>)	Removes instance <i>valueIdentifier</i> from the value(s) of property <i>propertyIdentifier</i> of instance <i>instanceIdentifier</i> . The statement applies both to functional and non-functional properties.	instanceIdentifier	String Literal Or DefaultOWLIndividual Variable	Input
		propertyIdentifier	String Literal	Input
		valueIdentifier	String Literal Or DefaultOWLIndividual Variable	Input
<i>getLocalName</i> (<i>instanceIdentifier</i> , <i>instanceName</i>)	Gets the local name (without the namespace prefix) of instance <i>instanceIdentifier</i> and stores it in variable <i>instanceName</i> .	instanceIdentifier	DefaultOWLIndividual Variable	Input
		instanceName	String Variable	Output
<i>setVariable</i> (<i>variableIdentifier</i> , <i>valueIdentifier</i>)	Sets the value of variable <i>variableIdentifier</i> to <i>valueIdentifier</i> .	variableIdentifier	String Literal	Input
		valueIdentifier	String Literal Or ANY Variable	Input

B6. ONTOMATH_TUTORING_PROCESSES_EDITING STATEMENTS

Statement	Purpose	Parameters		
		Name	Type	I/O
<i>addPerform</i> (<i>performStatementIdentifier</i>)	Adds a Perform Control Construct (tutoring statement) as a child to the currently selected construct in the displayed tutoring process tree	performStatementIdentifier	OWL-S Perform Control Construct Literal or Variable	Input

B7. ONTOMATH_ONTOLOGY_PREDICATES

Predicate	Purpose	Parameters		
		Name	Type	I/O
<i>authorConfirmation</i> (<i>message</i>)	Displays a Java Yes/No message box displaying <i>message</i> . If the author clicks on “Yes” it returns TRUE. If the author clicks on “No” it returns FALSE.	message	String Literal Or Variable	Input
<i>iteratorHasNext</i> (<i>iterator</i>)	Returns TRUE if <i>iterator</i> is non-empty. Otherwise it returns FALSE.	iterator	Iterator Variable	Input
<i>isNull</i> (<i>object</i>)	Returns TRUE if <i>object</i> is NULL. Otherwise it returns FALSE.	object	Object Variable	Input
<i>isNotNull</i> (<i>object</i>)	Returns TRUE if <i>object</i> is NOT NULL. Otherwise it returns FALSE.	object	Object Variable	Input

References

- Aitken, J. S., & Sklavakis, D. (1999). Integrating problem solving methods into CYC. In T. Dean (Ed.) *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)* (pp. 627-633). San Francisco: Morgan Kaufman Publishers.
- Aleven, V., McLaren, B., Sewall, J., & Koedinger, K. R. (2006). The Cognitive Tutor Authoring Tools (CTAT): Preliminary Evaluation of Efficiency Gains. *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS 2006)*, (pp 61-70). Berlin: Springer.
- Aleven, V., McLaren, B., Sewall, J., & Koedinger, K. R. (2009). A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. *International Journal of Artificial Intelligence in Education*, Vol. 19 (pp.105-154).
- Anderson, J.R. (1993). *Rules of the Mind*. Hillsdale, NJ: Erlbaum.
- Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *The Journal of the Learning Sciences*, Vol. 4 (2) (pp. 167-207).
- Blessing., B.S., Gilbert, B. S., Ourada, S. & Ritter S. (2009). Authoring Model-Tracing Cognitive Tutors. *International Journal of Artificial Intelligence in Education*, Vol. 19 (pp.189-210)
- Chandrasekaran, B. (1986). Generic Tasks for knowledge-based reasoning: the right level of abstraction for knowledge acquisition, *IEEE Expert*, Vol. 1 (pp. 23-30).
- Clancey, W. J. (1985). Heuristic classification, *Artificial Intelligence* Vol. 27 (pp. 289-350).
- Corbett, A. T. (2001). Cognitive Computer Tutors: Solving the Two-Sigma Problem. *Proceedings of the 8th International Conference on User Modeling 2001* (pp. 137-147). London: Springer.
- Crowley, R.S., & Medvedeva, O. (2006). An intelligent tutoring system for visual classification problem solving. *Artificial Intelligence in Medicine*, Vol. 36 (pp. 85-117).
- Denau, R., Dolbear, C., Hart, G., Dimitrova, V., & Cohn, A. (2011). Supporting Domain Experts to Construct Conceptual Ontologies: A Holistic Approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 9 (2) (pp. 113-127). Elsevier.
- Dicheva, D., Mizoguchi, R., & Greer, J. (Eds.) (2009). *Semantic Web Technologies for e-learning, The Future of Learning*, Vol. 4. Amsterdam: IOS Press.

-
- Gómez-Pérez, A., Fernández-López, M., & Corcho, O. (2004). *Ontological Engineering: With Examples from the Areas of Knowledge Management, E-commerce and the Semantic Web*. Berlin: Springer,
- Hoffman, R. (1987). The Problem of Extracting the Knowledge of Experts From the Perspective of Experimental Psychology. *AI Magazine* (pp. 53-67).
- Kaljurand, K. ACE View --- an ontology and rule editor based on Attempto Controlled English. *OWLED 2008*.
- Kingston, J. (1995). Applying KADS to KADS: knowledge-based guidance for knowledge engineering. *Expert Systems*. Vol. 12(1), 15-26.
- Koedinger, K. R., Anderson, J.R., Hadley, W.H., & Mark, M. A. (1997). Intelligent Tutoring Goes to School in the Big City, *International Journal of Artificial Intelligence in Education*, Vol. 8 (pp. 30-43).
- Koedinger, K. R., & Corbett, A. (2006). Cognitive Tutors: Technology bringing learning science to the classroom. In K. Sawyer (Ed.) *The Cambridge Handbook of the Learning Sciences* (pp. 61-78), Cambridge University Press.
- Lenat, D. B., & Guha, R. V. (1990). Building large Knowledge-based systems. Representation and inference in the Cyc project. Reading, Massachusetts: Addison-Wesley.
- Lenat, D. B. (1995). CYC: A Large-Scale Investment in Knowledge Infrastructure. *Communications of the ACM*, 38 (11).
- Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., & Sycara, K. (2005). Bringing Semantics to Web Services: The OWL-S Approach, LNCS Vol. 3387 (pp. 26-42). Berlin: Springer.
- Mellis, E., Andrès, J., Büdenbender, J., Frischauf, A., Gogvadze, G., Libbrecht, P., Pollet, M. & Ullrich, C. (2001). A Generic and Adaptive Web-Based Learning Environment. *International Journal of Artificial Intelligence in Education*, 12, 385-407.
- Mitrovic, A., Koedinger, K., & Martin, B. (2003). A Comparative Analysis of Cognitive Tutoring and Constraint-Based Modelling. In P. Brusilovsky, A. Corbett & F. de Rosis (Eds.) *Proceedings of the 9th International Conference on User Modeling UM 2003* (pp. 313-322). LNAI 2702. Berlin: Springer-Verlag.
- Mitrovich, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., & Holland, J. (2009). ASPIRE: An Authoring System and Deployment Environment for Constraint-Based Tutors. *International Journal of Artificial Intelligence in Education*, Vol. 19 (pp. 155-188)

- Mizoguchi, R., Vankwelkenheusen, J., & Ikeda, M. (1995). Task Ontology for Reuse of Problem Solving Knowledge. *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing* (pp.46-59). Netherlands: IOS Press.
- Mizoguchi, R., & Bourdeau, J. (2000). Using ontological engineering to overcome common AI-ED problems. *International Journal of Artificial Intelligence in Education*, Vol. 11 (pp. 107-121).
- Mizoguchi, R. (2004). Tutorial on ontological engineering: part 3: Advanced course of ontological engineering. *New Generation Computing*. Vol. 22 (2) (pp. 198-220). Berlin: Springer.
- Mizoguchi, R., Hayasi, Y., & Bourdeau, J. (2009). Inside a Theory-Aware Authoring System. In D. Dicheva, R. Mizoguchi & J. Greer (Eds.) *Semantic Web Technologies for e-learning: The Future of learning*, Vol. 4 (pp. 59-76). Amsterdam: IOS Press.
- Murray, T. (2003a). An overview of intelligent tutoring system authoring tools: Updated Analysis of the State of the Art. In T. Murray, S. Ainsworth, & S. Blessing (Eds.) *Authoring Tools for Advanced Technology Learning Environments* (pp 491-544). Netherlands: Kluwer Academic Publishers.
- Murray, T. (2003b). Principles for Pedagogy-Oriented Knowledge Based Tutor Authoring Systems. In T. Murray, S. Ainsworth, & S. Blessing (Eds.) *Authoring Tools for Advanced Technology Learning Environments* (pp 439-466). Netherlands: Kluwer Academic Publishers.
- Paquette, L., Lebeau, J.-F., & Mayers, A. (2010). Authoring Problem-Solving Tutors: A Comparison Between CTAT and ASTUS. In Nkambou, R., Bourdeau, J., & Mizoguchi, R. (Eds.) *Advances in Intelligent Tutoring Systems* (pp 377-405). Heidelberg: Springer.
- Puerta, A. R., & Musen, M. (1992). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tasks. *Knowledge Acquisition*, Vol. 4 (pp. 171-196).
- Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., Van de Velde, W., & Wielinga, B. (1999). *Knowledge Engineering and Management: The CommonKADS Methodology*. Cambridge MA: MIT Press.
- Sklavakis, D. (1998). Implementing Problem Solving Methods in CYC. *MSc Dissertation*, Department of Artificial Intelligence, University of Edinburgh.
- Sklavakis, D., & Refanidis, I. (2008). An Individualized Web-Based Algebra Tutor Based on Dynamic Deep Model-Tracing. Proceedings of the Fifth Hellenic Conference on Artificial Intelligence (SETN '08), (pp. 389-394). Heidelberg: Springer.

-
- Sklavakis, D. & Refanidis, I. (2009a). The MATHESIS Algebra Tutor: Web-based Expert Tutoring via Deep Model Tracing. Interactive Event. Proceedings of the 14th International Conference on Artificial Intelligence in Education (AIED 2009), (p. 795). Amsterdam: IOS Press.
- Sklavakis, D., & Refanidis, I. (2009b). The MATHESIS Ontology: Reusable Authoring Knowledge for Reusable Intelligent Tutors. Proceedings of the 7th International Workshop on Ontologies and Semantic Web for E-Learning (SWEL 2009), (pp. 86-90).
- Sklavakis, D., & Refanidis, I. (2010a). MATHESIS: A Web-Based Intelligent Tutoring School for Algebra. Intelligent System Demonstration at the 6th Hellenic Conference on Artificial Intelligence (SETN 2010).
- Sklavakis, D., & Refanidis, I. (2010b). Ontology-Based Authoring of Intelligent Model-Tracing Math Tutors. Proceedings of the Fourteenth International Conference on Artificial Intelligence (AIMSA 2010), (pp. 201-210). Heidelberg: Springer.
- Sklavakis, D., & Refanidis, I. (2013). MATHESIS: An Intelligent Web-Based Algebra Tutoring School. *International Journal of Artificial Intelligence in Education* Vol. 22 (2) (pp. 191-218). Amsterdam: IOS Press.
- Sklavakis, D., & Refanidis, I. (2014). The MATHESIS meta-knowledge engineering framework: Ontology-driven development of intelligent tutoring systems. *Applied Ontology* Vol. 9 (3-4) (pp. 237-265). Amsterdam: IOS Press.
- Suraweera, P., Mitrovic, A., Martin, B., Holland, J., Milik, N., Zakharov, K., & McGuigan, N. (2009). Using Ontologies to Author Constraint-Based Intelligent Tutoring Systems. In D. Dicheva, R. Mizoguchi & J. Greer (Eds.) *Semantic Web Technologies for e-learning: The Future of learning*, Vol. 4 (pp. 59-76). Amsterdam: IOS Press.
- Tudorache, T., Nyulas, C., Noy, N.F., & Musen, M.A. (2013). WebProtégé: A Distributed Ontology Editor and Knowledge Acquisition Tool for the Web. *Semantic Web* Vol. 4 (1) (pp. 89-99). Amsterdam: IOS Press.
- VanLehn, K., Jones, R. M., & Chi, M. T. H. (1992). A model of the self-explanation effect. *Journal of the Learning Sciences*, 2(1), 1-59.
- VanLehn, K. (1999). Rule learning events in the acquisition of a complex skill: An evaluation of Cascade. *Journal of the Learning Sciences*, 8(2), 179-221.
- VanLehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., & Wintersgill, M. (2005). The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education* Vol. 15(pp. 147-204). Amsterdam: IOS Press.

- VanLehn, K. (2006). The Behavior of Tutoring Systems. *International Journal of Artificial Intelligence in Education* Vol. 16 (3) (pp. 227-265). Amsterdam: IOS Press.
- Wielinga, B.J., Schreiber, A.Th., & Breuker, J.A. (1992). KADS: A modeling approach to knowledge engineering. *Knowledge Acquisition* Vol. 4 (1) (pp. 5-53). Elsevier