Deliverable N° D24

Tutorial Component

The LeActiveMath Consortium
March 2006

Main Authors: Carsten Ullrich, Eric Andrès, Philipp Kärger, Erica Melis, Marianne Moormann

Project funded by the European Community under the Sixth Framework Programme for Research and Technological Development
<table>
<thead>
<tr>
<th><strong>Project ref.no.</strong></th>
<th>IST-507826</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project title</strong></td>
<td>LEActiveMath - Language-Enhanced, User Adaptive, Interactive e-learning for Mathematics</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Deliverable status</strong></th>
<th>Restricted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Contractual date of delivery</strong></td>
<td>Project Month 27 – 2006 March 31st</td>
</tr>
<tr>
<td><strong>Actual date of delivery</strong></td>
<td>2006 May 15th</td>
</tr>
<tr>
<td><strong>Deliverable title</strong></td>
<td>Tutorial Component</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Prototype</td>
</tr>
<tr>
<td><strong>Status &amp; version</strong></td>
<td>1.25</td>
</tr>
<tr>
<td><strong>Number of pages</strong></td>
<td>143</td>
</tr>
<tr>
<td><strong>WP contributing to the deliverable</strong></td>
<td>WP3</td>
</tr>
<tr>
<td><strong>WP/Task responsible</strong></td>
<td>DFKI / Carsten Ullrich</td>
</tr>
<tr>
<td><strong>Author(s)</strong></td>
<td>Carsten Ullrich, Eric Andrés, Philipp Kärger, Erica Melis, Marianne Moormann</td>
</tr>
<tr>
<td><strong>EC Project Officer</strong></td>
<td>Colin Stewart</td>
</tr>
<tr>
<td><strong>Keywords</strong></td>
<td></td>
</tr>
</tbody>
</table>
Contents

Executive Summary 6

1 Introduction 7
  1.1 Relevant Parts from the Description of Work and Requirements Analysis 7
  1.2 Links with other Deliverables 12
  1.3 Design Decisions based on the Pedagogical Principles 12
    1.3.1 Moderate Constructivism 13
    1.3.2 Competency-based Learning 13
    1.3.3 Consequences on System Design 13
  1.4 Challenges 14

2 Tutorial Component 16
  2.1 Overview of the Architecture 16
  2.2 A Declarative, Task-Based Representation of Pedagogical Knowledge 19
    2.2.1 Pedagogical Tasks 19
    2.2.2 Pedagogical Methods 20
  2.3 Ontology of Instructional Objects 20
  2.4 Course Generator 24
    2.4.1 Hierarchical Task Network Planning 24
      2.4.1.1 Problem Solving in HTN-Planning 24
      2.4.1.2 Challenges of HTN Planning in Distributed Information Systems 25
    2.4.2 A Mediator for Repository Integration 27
      2.4.2.1 Queries and Query Language 28
      2.4.2.2 Ontology Mapping and Query Rewriting 28
      2.4.2.3 Architecture 30
      2.4.2.4 Wrappers 31
      2.4.2.5 Caching 31
    2.4.3 Representing Course Generation Knowledge in a HTN-planner 31
2.4.3.1 Implementation of the Scenario LearnNew ............... 31
2.4.3.2 Providing Structure and Adaptivity: Dynamic Tasks ........ 42
2.4.3.3 Converting a Plan into a Table of Contents ............... 42
2.4.3.4 Distinguishing Between Dynamic and Stable Information .... 44
2.4.3.5 Integration of the Course Generator ...................... 45
2.4.3.6 Summary .................................................. 45

2.5 Supporting the User’s Initiative ................................. 47

2.6 Exercise Sequencer .............................................. 48
2.6.1 Using Competency Levels for Exercise Sequencing .......... 48

2.7 Suggestion Agent ................................................ 50
2.7.1 Local and Global Feedback .................................. 50
2.7.2 Global Feedback in LeActivemath ........................... 51
2.7.2.1 Architecture ............................................. 52
2.7.2.2 Diagnostic Process ...................................... 54
2.7.2.3 Suggestion Reasoner ..................................... 56
2.7.2.4 Conflict Management .................................... 58
2.7.2.5 The GUI and the Suggestion Rendering ................... 58

2.8 Tutorial Control: Managing other Components Access to the Course Generator ........................................ 59
2.8.1 Concurrent Access of LeActiveMath Components to the Course Generator ........................................ 59
2.8.2 Context-Dependent Access of LeActiveMath Components to the Course Generator .......................... 60

2.9 Summary .......................................................... 60

3 Integration into LeActiveMath ........................................ 61
3.1 General Integration ............................................... 61
3.1.1 Interfaces of the Tutorial Control ........................... 61
3.1.2 Interfaces of the Course Generator .......................... 62
3.1.3 Events ......................................................... 63
3.1.4 Public Pedagogical Tasks .................................... 63
3.1.5 Dynamically Generated Elements in a Table of Contents ... 65
3.1.6 A Generic Integration of Learning Supporting Services ... 65

3.2 Integration of LeActiveMath Components ......................... 66
3.2.1 Extended Learner Model ..................................... 66
3.2.2 Situational Model ........................................... 67
3.2.3 Local Tutorial Dialogue ...................................... 67
3.2.4 Interactive Concept Mapping Tool .......................... 67
3.2.5 Open Learner Model

3.2.5.1 The Open Learner Model as a Learning Supporting Service

3.2.5.2 Suggestions to the Tutorial Component

3.2.6 Assembly Tool

3.2.7 Symbolic Representations for Text Generation

4 Evaluations

4.1 Evaluations of the Ontology Of Instructional Objects

4.2 Technical Evaluations of the Tutorial Component

4.2.1 Performance of the Course Generator

4.2.2 Performance of the Mediator

4.3 First Formative Evaluations in Saarbrücken and Munich

4.4 Lab-based Evaluation in Edinburgh

4.5 Further Planned Evaluations

5 Conclusion

5.1 Related Work

5.2 Outstanding Issues

A Interfaces and Events

B Planning Domain Description
Executive Summary

This report provides an account of Deliverable D24, Tutorial Component, and contains a description of the architecture and the implementation of the Tutorial Component, and its integration into the LeActiveMath system. It includes an explanation of the technical and pedagogical principles underlying the design and implementation of the Tutorial Component, and a description of the actual implementation.

The Tutorial Component realizes the adaptivity, reactivity, and integration of tools: it selects and assembles learning material adaptively relative to the learner’s goal, competencies and preferences; reacts by monitoring the learner’s interactions with the LeActiveMath system and providing supportive feedback and it responds to requests of the learner; it integrates learning supporting tools developed in LeActiveMath by seamlessly blending learning materials and the tools in courses presented to the learner.

The Tutorial Component consists of three main sub-components, which each serves a specific functionality: a Course Generator adaptively selects learning material that supports learners in achieving pedagogical tasks; a Suggestion Agent monitors a learner’s behavior and provides remediating feedback in case potential problems are detected; an Exercise Sequencer adaptively selects exercises leading the learner towards a higher competency level.

The Tutorial Component uses formalized pedagogical knowledge for its decision making. The deliverable D20, “Formalized Pedagogical Strategies”, served to make the often implicit pedagogical expert knowledge explicit. The result was a collection of rules of how to assemble learning material such that it supports LeActiveMath pedagogical principles: the implemented knowledge follows a moderate constructivist learning paradigm, which emphasizes the active role of the learner in the learning process, but does not neglect support and guidance. These principles are reflected in the overall design of the Tutorial Component. The learner is in control but supported both on request and automatically.

For this Tutorial Component, the rules were implemented using Artificial Intelligence planning techniques. The selected framework, Hierarchical Task Network Planning, has proven its efficiency in several real-world applications and is well suited for representing human expert knowledge due to its hierarchical nature.
Chapter 1

Introduction

The goal of the LeActiveMath project is to design an intelligent Web-based e-learning system for mathematics. One of its major features is adaptivity. The mathematical course material presented to the learner is adapted to the learner’s goals, the learning scenario, the learner’s competency-level, and his individual preferences. Furthermore, LeActiveMath integrates tools that can be used in exercises and for exploratory learning.

The Tutorial Component is the central component for the adaptivity of LeActiveMath and the provision of moderate constructivist learning opportunities. The adaptive features are motivated by pedagogical and cognitive research. In particular, LeActiveMath is learner-centered and supports the learner’s initiative. The technology as well as the content that is developed to evaluate the technology realizes a moderate constructivist and problem-based approach to learning and teaching mathematics.

In this chapter, we will describe the factors that influenced the design of the Tutorial Component. We start with the requirements raised by the description of work and the claims collected in the requirement analysis. In Section 1.2, we put the Tutorial Component in context with the results of the other workpackages. The LeActiveMath project emphasizes the relevance of pedagogy: the best possible technical tools will not enhance learning unless based on well-founded pedagogical results. Section 1.3 therefore lays out the pedagogical principles the Tutorial Component is based on. The final section of this chapter summarizes the challenges that arise from the technological as well as pedagogical requirements.

1.1 Relevant Parts from the Description of Work and Requirements Analysis

In this section, we list the relevant sections from the description of work [38] and the requirement analysis [41], and describe how they were met.

**T3.4 Reactive tutorial component.** This component will realize the adaptation to the learner and the realization of tutorial strategies. The tutorial component will generate personalized static material to begin with as well as global suggestions on what could be useful to consider, to search, to organize, to learn next, how to navigate, etc. according to the student’s activities. It will compute tutorial/learning goals that can also be used by other components, e.g., the dialogue component. Hence, the tutorial component has to provide an interface for the tutorial dialog components. As we learned from our previous experience, this is a non-trivial task. Among the difficult research problems are: (1) react to diagnoses of the learner’s activities and state may come from different...
sources, including exercise assessment, other action diagnoses, motivational diagnosis have to be jointly considered. Basic research is still needed in order to be able to compute consequences from the diagnoses according to cognitive and pedagogical results. (2) Not just the content and its sequencing have to be adapted to the student’s needs, goals and capabilities but also the multi-modal delivery. (3) How to formalize and represent tutorial goals that restrict or enable navigation? etc.

D5, Requirement Analysis, served to further specify these general requirements. The consortium of the LEACTIVEMATH project adopted Carroll’s Scenario-Based Design and Claims Analysis methodology [7] which resulted in the following claims covering the Tutorial Component:

<table>
<thead>
<tr>
<th>Requirement 4.12</th>
<th>The tutorial component (TC) allows to represent and execute different pedagogical strategies, e.g. problem-based, traditional didactical approach.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>Different ways of teaching/learning.</td>
</tr>
<tr>
<td>Because</td>
<td>Different ways of teaching use different pedagogical strategies.</td>
</tr>
<tr>
<td>Check-rule</td>
<td>Empirical test with small groups using different strategies.</td>
</tr>
<tr>
<td>Issues</td>
<td>• Long-term effects are hard to evaluate.</td>
</tr>
<tr>
<td></td>
<td>• Pedagogical strategies can become very complex and hard to formalize</td>
</tr>
<tr>
<td></td>
<td>• Large collections of learning objects are required.</td>
</tr>
</tbody>
</table>

Claim 4.12 has been met by the implementation of different pedagogical scenarios, which is described in detail in Section 2.4. Deliverable D20 (Formalized Pedagogical Strategies) [47] specified six different scenarios which were implemented for this deliverable. These scenarios are based on a mildly constructivist learning theory (one of the basic pedagogical principles in LEACTIVEMATH, see the following section). An additional scenario that is based on more traditional didactical approach is guidedTour. This scenario was implemented already in a former version of the course generator and re-implemented in the current version. Therefore, we can safely claim that strategies based on different pedagogical principles can be realized with the current tutorial component. Additionally, the re-implementation, although outside the scope of LEACTIVEMATH, served to compare the performance of the former and current version of the course generator (see Section 4.2.1).
<table>
<thead>
<tr>
<th>Requirement 4.13</th>
<th>Allow for reactivity: tutorial component should react to learner’s progress and problems.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>Individual needs of the student.</td>
</tr>
<tr>
<td>Because</td>
<td>Course generation (which happens before the student accesses the course) may be based on assumptions that change during learning.</td>
</tr>
<tr>
<td>Check-rule</td>
<td>Test with small group of students (reactivity vs. non-reactivity).</td>
</tr>
<tr>
<td>Issues</td>
<td>- Changes to the course have to be indicated to the learner.</td>
</tr>
<tr>
<td></td>
<td>- Reactivity may create navigation problems.</td>
</tr>
</tbody>
</table>

Reactivity as demanded by claim 4.13 is an inherent feature of the Tutorial Component and has been met by several features and components: a suggestion agent monitors the navigation and problem-solving behavior of the student and reacts by providing different kind of hints (described in Section 2.6); an exercise sequencer dynamically selects exercises based on the current learning goal and performance of the learner (see Section 2.6); the extension of static tables of contents with dynamic elements that trigger the on-demand selection of learning objects allows to account for changes of the learners’s competency levels that take place in the time between course generation and possibly much later content browsing (Section 2.4.3.2).

<table>
<thead>
<tr>
<th>Requirement 4.14</th>
<th>Allow for interactivity: student should be able to ask (actively) for specific additional content and integrate it into her course.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Because</td>
<td>Active requesting.</td>
</tr>
<tr>
<td>Check-rule</td>
<td>Test: do learners use the feature?</td>
</tr>
<tr>
<td>Issues</td>
<td>- Maybe this feature is appropriate only in some pedagogical scenarios.</td>
</tr>
<tr>
<td></td>
<td>- Interactivity may create navigation problems.</td>
</tr>
<tr>
<td></td>
<td>- Do we need NLG for this feature?</td>
</tr>
</tbody>
</table>

This claim is closely related to the previous one: automatic reactions to the learner’s progress rely on heuristics and hence potentially may fail. To allow the learner to ask actively for content on the one hand can compensate incorrect suggestions and on the other hand supports the active engagement of the learner aimed at by moderate constructivism. Section 2.5 describes how the learner can access the pedagogical knowledge of the Tutorial Component and use it to modify generated courses. The second, even more important feature contributing to interactivity is the
design of access to the course generation: learners actively select the type of course (scenario) and content, and thereby learn to become aware of and articulate their learning goals.

<table>
<thead>
<tr>
<th>Requirement 4.15</th>
<th>Need ontology of instructional objects.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>• Integration of third-party content.</td>
</tr>
<tr>
<td></td>
<td>• Provides adequate level of abstraction for authors/learners to talk about learning materials.</td>
</tr>
<tr>
<td>Because</td>
<td>There exist many knowledge representations of e-learning content and using a shared ontology supports integration. Users do not want to know about internal knowledge representations.</td>
</tr>
<tr>
<td>Check-rule</td>
<td>Check: can the learning material be described by the ontology? Do users understand the ontology?</td>
</tr>
<tr>
<td>Issues</td>
<td>• Semantics of ontologies is partly not properly defined.</td>
</tr>
<tr>
<td></td>
<td>• Mapping of ontologies may be ambiguous.</td>
</tr>
</tbody>
</table>

Claim 4.15 aims at easing re-usability of content and pedagogical knowledge. For that purpose, we developed an ontology that provides a vocabulary to describe the instructional purpose of learning materials (Section 2.3). It is used in the implementation of the pedagogical strategies specified in D20. Therefore, the strategies do not depend on a proprietary knowledge representation used in a specific learning repository, but can be applied in other contexts as well. The mediator architecture described in Section 2.4.2 serves as the technological basis for re-usability and allows to connect the course generator to third-party learning repositories.

<table>
<thead>
<tr>
<th>Requirement 4.16</th>
<th>TC generates symbolic representations for transitions, introductions, and summarisations that the NLG front-end then verbalises into English (WP5).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>Better readability and coherence.</td>
</tr>
<tr>
<td>Because</td>
<td>Transition and summarising texts make the transition between the learning objects smoother.</td>
</tr>
<tr>
<td>Check-rule</td>
<td>Test effect with small group.</td>
</tr>
<tr>
<td>Issues</td>
<td>Canned text can be too boring.</td>
</tr>
</tbody>
</table>

This claim has been met by the techniques described in Section 3.2.7, which allow the course generator to create symbolic representations of texts. Although, the task of rendering the representations has been moved from Edinburgh to Saarland University, a preliminary rendering engine is already implemented and provides evidence that the symbolic representations contain sufficient information for text generation.
<table>
<thead>
<tr>
<th>Requirement 4.17</th>
<th>The TC offers scenarios that target competencies.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>Competency-level pedagogy as used in PISA and other studies.</td>
</tr>
<tr>
<td>Because</td>
<td><strong>LeActiveMath</strong> should be able to conform with didactical standards.</td>
</tr>
<tr>
<td>Check-rule</td>
<td>Compare results with national curricula and courses that comply with such standards.</td>
</tr>
</tbody>
</table>
| Issues           | • Formalisation and encoding of competency standards.  
|                  | • Needs appropriate content.  
|                  | • Variation in different European countries (has to be discovered by Augsburg in the first place). |

This claim has been met by the implementation of the pedagogical strategies that were described in D20 [47] (Section 2.4.3). As a result, competencies and competency levels, which today dominate modern pedagogical theories, are supported throughout the complete Tutorial Component: the examples and exercises in each scenario are selected from the whole range of competencies and correspond to the learner’s actual competency level. To our knowledge, this is a unique feature and makes **LeActiveMath** the first e-learning environment that supports competency-based pedagogy starting with the metadata of the learning objects, over learner modeling to adaptive content selection.

<table>
<thead>
<tr>
<th>Requirement 4.18</th>
<th>The TC needs to have access to all instructionally relevant information about the user.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>Adaptivity.</td>
</tr>
<tr>
<td>Because</td>
<td>Without information about the user, the TC cannot provide adequate adaptations.</td>
</tr>
<tr>
<td>Check-rule</td>
<td></td>
</tr>
</tbody>
</table>
| Issues           | • The relevant information includes the learner’s current knowledge state, her preferences, her history, her learning/cognitive style, misconceptions, static profile and traits.  
|                  | • We may need for an ontology similar to the ontology of instructional objects in case we want to switch between different learner models that use different representations of individual information. |

The learner model is an essential part of **LeActiveMath** and has been described in several deliverables ([51, 45, 50]. The Tutorial Component makes extensive use of the information provided by the eXtended Learner Model (xLM), especially the Course Generator (Section 2.4). The formalized pedagogical strategies of D20 provide the basis of content selection according to learner information. Section 3.2.1 presents the used interfaces.
1.2 Links with other Deliverables

The Tutorial Component is one of the central components of LeActiveMath and hence depends on the outcome of several other deliverables. Structured according to the workpackages, the most relevant are:

**WP1** D5 (Requirement Analysis, [41]) contains the claims generated during the requirement analysis. As described in the previous section, the claims served as fundamental criteria for the design of the Tutorial Component.

**WP2** D6 (Structures and Metadata Model, [42]) specified the metadata used to characterize the individual learning objects in LeActiveMath. Obviously, the available metadata affects the pedagogical knowledge that reasons about the learning objects. Nevertheless, in Chapter 2 we will see that the Tutorial Component is to a certain degree independent of the concrete knowledge representation used in LeActiveMath.

**WP3** D8 (Open Architecture, [39]) describes the basic architectural guidelines of LeActiveMath such as modularity, loose coupling, event-based communication, and a model-view-controller framework. The Tutorial Component complies to these requirements as the description of its architecture in Chapter 2 and of the integration into the LeActiveMath system (Chapter 3) will show.

The integration of the tools developed in WP3 is an important part of the Tutorial Component, too. Chapter 3 describes the general integration approach as well as the specific details for the deliverables D28 (Interactive Concept Map) and D37 (Assembly Tool).

**WP4** D10 (Specification Student Model, [51]) and the related deliverables (D29 and D30, [50, 45]) are crucial for individualized adaptation. Without proper modeling of learner information, personalization is impossible. The integration of the Tutorial Component and the xLM is described in Chapter 3.

**WP5** D17 (Local Tutorial Dialogue, [49]) is a dialogue system that is used to support interactive exercises in a flexible and user-adaptive fashion. It communicates with the Tutorial Component like any other tool in LeActiveMath using the mechanisms described in Chapter 3.

**WP6** D18/D19 (Content, [43, 44]) are essential for the Tutorial Component: without sufficiently varying content, there is nothing to use for adaptivity. Therefore, DFKI and Augsburg / Munich, the respective WP-leaders, closely communicated almost daily to ensure that both the development of the Tutorial Component and the content were aligned properly. Similar, if not more important was D20 (Formalized Pedagogical Strategies, [47]), the basis for the pedagogical knowledge implemented in the Tutorial Component. Again, both deliverables D20 and D24 benefited from the close cooperation between DFKI and Augsburg/Munich.

1.3 Design Decisions based on the Pedagogical Principles

The goal of the LeActiveMath consortium is to advance the state of the art in e-learning by applying novel technologies to modern pedagogical theories. We deliberately decided to base the pedagogical principles on moderate constructivist learning theory as well as to pursue a competency-based approach. In this section (adapted from D20, [47]), we will describe how these principles influenced the design of the LeActiveMath system, especially with regard to the Tutorial Component.
1.3.1 Moderate Constructivism

LeActiveMath is based on moderate constructivist learning theory. Relying on modern brain and mind research, these theories claim that learning processes can only be initialized by learners themselves, by their active (constructivist) operation on the subject domain. Hence, learners play an active role in their learning processes and are responsible for the outcome of their learning processes to a major extent.

Therefore the LeActiveMath strategies do not implement the traditional knowledge transfer paradigm but aim at supporting the students in structuring their learning activities and developing strategic competence. In other words, the strategies serve the overall goal that students become autonomous and self-regulated learners.

However, in a mildly constructivist paradigm there is still need for support and structure. Studies, e.g., [69, 66], show that in particular low-achieving students may benefit from content organized according to pedagogical principles.

Pre-structured content facilitates access to content. Without it, students who are unable to structure their learning process might not be able to improve their mathematical abilities at all. The usage of pedagogical strategies to provide learning paths through the content allows this group to gain knowledge and learning capabilities.

Additionally, automatically generated courses provide examples of how to deal with the content available in LeActiveMath. They show how the material can be structured or ordered depending on the educational objectives. Because the learner is able to select between different scenarios, he becomes aware of his learning goals and trains to articulate them.

It is important to note that although the content is pre-structured, the learner is never constrained her navigation. He can freely browse through a course and has full access to the complete content available via the search facility. Placing the locus of control in the hands of the learner is one of the few measures with respect to e-learning that has repeatedly proved to increase motivation (for a meta-review, see [76], p.151-162).

1.3.2 Competency-based Learning

The competency approach is based on the literacy concept. The general assumption is that different competencies together build up mathematical literacy. One can only become mathematically literate by sufficiently high achievement over the whole set of competencies.

The pedagogical claim is that students make better learning progress if they interact with different perspectives on concepts. The competency approach can be considered as a way to support the presentation of concepts from different perspectives by giving varying tasks to the students. The tasks differ in the required mathematical activities, the competencies.

The competencies in LeActiveMath describe higher level learning objectives, which are based on current international discussions in mathematics education, with strong influence by the Program for International Student Assessment (PISA) of the OECD and NCTM (the National Council of Teachers of Mathematics). An advantage besides being conform to and contributing to the contemporary developments in mathematics education are that that these competencies are (almost) not age- or content-specific and their international validity.

1.3.3 Consequences on System Design

The pedagogical principles underlying LeActiveMath have several consequences on the technological design of the overall system and the Tutorial Component in particular. Competency-based
learning is best reflected in the pedagogical scenarios defined in D20 [47], with the implementation described in Section 2.4.

Moderate constructivism influences the complete system design, especially navigation. In LEACTIVEMATH, the learner is never restricted in his navigation. She can access any page in a course and return to the main menu at any time. This is in contrast to a number of existing e-learning systems, especially those that implement the IMS Simple Sequencing Standard (IMS SS, [30]).

Using IMS SS, an author specifies the structure (i.e., sections and subsections) of a collection of learning materials and provides information on how to guide the learner through this structure. Conditional rules can make, for instance, a learner skip a section if her knowledge exceeds a given threshold, or inhibit access to a section.

The pedagogical approach of IMS SS follows a more traditional school of pedagogy, where strict control over student is esteemed to be necessary. In our approach, such strict restrictions are not required. Additionally, an IMS SS runtime environment that implements the standard is a complex software that imposes requirements on the overall system. Therefore we decided against using IMS SS.

An open system as developed in LEACTIVEMATH has the additional advantage that it is in line with the expectations of a Web-user. A browser can restrict the navigation only in a very limited way, and especially back/forward movements through the hyperlink space using the browser button can not controlled at all.

Still, the context of a learning session, that is, the selected scenario, influence the behavior of several components. For instance, during an exam simulation, the feedback given by the exercise system is restricted: hints and supportive feedback are deactivated. But again, complete control over the learners’ access to tools etc. can not be achieved, and, in line with the pedagogical principles is not necessary.

1.4 Challenges

Difficulties that we had to solve for the design and implementation of the arise from a pedagogical as well as a technical side. The following list summarizes the challenges we encountered:

Complex knowledge. As the description of the pedagogical scenarios in D20 [47] shows, pedagogical decision-making requires taking into account complex information about the learner and the available content. The framework chosen for implementing the knowledge needs to be able to represent this information. Additionally, if possible, the framework should take advantage of the hierarchical structure inherent in the pedagogical scenarios.

Automatic processing. The learner can define her learning goals on her own. Hence, it is practically impossible to pre-calculate all potential combinations and to check them manually by a human tutor. Therefore, the execution of the pedagogical knowledge needs to happen automatically. Among others, this requires access to sufficiently expressive metadata about the learning objects. As we will see in the Section 2.3, existing standards are not sufficient for that purpose.

Re-usability of the pedagogical strategies. Designing pedagogical strategies is a resource-intensive process. Therefore, the strategies should be as independent from the actual content as possible. This rules out approaches as described by Vassileva [88] where the learning objects themselves are annotated with pedagogical knowledge. Ideally, even though the strategies described in the previous section were developed in the LEACTIVEMATH project that targets calculus content, they should also be applicable to, e.g., fractions.
Distributed content. In Web-based environments, learning objects may be distributed over several repositories and annotated with different metadata schemas. Although this is not the case if one considers the LeActiveMath environment as a single system, with its well-defined repositories, today in the time of Web-services such an exclusive view would be too narrow. Hence, in order to make its services available to other, third-party systems, it is important that the Tutorial Component can abstract from the concrete knowledge sources. It should not need to locate the relevant sources nor should it interact with each source separately.

Efficient processing. Course generation involves querying user information, requests to content repositories, and processing the pedagogical knowledge itself. But as the courses are to be generated upon the learners’ requests, the assembly needs to happen within an acceptable delay.

Integration of learning supporting services. A vast range of services that support the learning process in various ways have been developed. The Tutorial Component should integrate these services, not arbitrarily but in a pedagogically sensible way: during the learning process, at specific times the usage of a tool will be more beneficial than at some other time. For instance, reflecting over the learned content may be most effective at the end of a lesson. Additionally, access to these services may vary, depending on the configuration of the learning environment and their general availability. Therefore, the course generation needs to take into account dynamic information about these services.

In the next chapters, we will describe how these challenges were met. We will start with a description of the Tutorial Component, then show how it is integrated in the LeActiveMath system.
Chapter 2

Tutorial Component

2.1 Overview of the Architecture

The Tutorial Component serves as an umbrella that collects several separate components and administers access to these components in order to realize a coherent pedagogical behavior of the LeActiveMath system.

Thus, the learner does not interact directly with the Tutorial Component in a way he does with other components, e.g., the Open Learner Model that provides a graphical user-interface. Instead, his requests and actions are passed to sub-components of the Tutorial Component, which then decide how to react to them.

Figure 2.1 illustrates the use cases involving the Tutorial Component. In this figure, a human user interacts with the sub-components of the Tutorial Component in two ways: by obtaining suggestions and by requesting content. The former case covers reactivity (see claim 4.13): depending on the learner’s progress the system can support him by providing suggestions, either learning material to work with or navigational hints (see Section 2.7). Additionally, the system can provide him with a sequence of exercises that leads him towards a learning goal (see Section 2.6).

The latter case consists of the learner actively asking for learning material, either a complete course or one or several learning objects (see claim 4.14). This use case is also used by clients, i.e., other LeActiveMath or even third-party components. These use cases are handled by the Course Generator described in Section 2.4.

The Tutorial Component breaks down into several separate components and interfaces. The diagram in Figure 2.2 shows their dependencies to other relevant LeActiveMath and third-party components. The following list briefly describes the different components:

- the Course Generator dynamically assembles a sequence of learning materials that fulfills given learning goals. These goals, called pedagogical tasks are central to the Tutorial Component and will be discussed in detail in Section 2.2. The course generator offers its functionality to other components via the interface AchieveTask. During the assembly, it uses information about the learner and the available content and hence queries the xLM, the LeActiveMath repositories and potentially other available third-party repositories. Once a course is assembled, it is passed to the presentation architecture [40], which manages the display to the learner. Section 2.4 describes the Course Generator in detail.

1 In Figure 2.1 and in the following figures, we will use UML notation.
2 In Figure 2.2, this functionality is represented by the ball-and-socket icon. The ball represents a provided interface, the socket a required interface.
• The **Exercise Sequencer** provides the learner with a sequence of dynamically selected exercises that leads him towards a learning goal. The sequencer uses the Course Generator to select adequate exercise. It is described in Section 2.6.

• The **Tutorial Control** manages access to the Course Generator by providing the interface TaskHandling. Depending on the user’s learning context, the usage of the Course Generator might be restricted. For instance, in an exam simulation, neither the Suggestion Agent nor the learner himself should be able to use the services of the Course Generator. The Tutorial Control is described in Section 2.8.

• The **Suggestion Agent** monitors the user’s interactions and offers feedback in case problems are diagnosed. Suggestions consists of navigational hints or learning materials. For the latter case, the Suggestion Agent uses course generation but controlled by the Tutorial Component. This component is described in Section 2.7.

• The **learner interface** allows the learner to ask actively for content by using the pedagogical knowledge contained in the Course Generator. Again, access is controlled by the Tutorial Component. The interface is described in Section 2.5.

Other **LeActiveMath** components can access the Course Generator as well. For instance, the xLM might use an exercise in case an impasse is reached during the negotiation with the learner. A typical flow of behavior in the Tutorial Component is illustrated in the activity flow chart displayed in Figure 2.3. Some component, (say the Suggestion Agent), sends a task to the Tutorial Component. The Tutorial Component verifies whether the task can be processed in the current learning context and, if it can, forwards it to the Course Generator. The Course Generator then tries to achieve the task. The resulting elements are presented later on. In case the task can not be processed or achieved, a failure indication is returned.
Figure 2.2: The Tutorial Component and dependencies on other LEActiveMath and third-party components

Figure 2.3: A typical activity flow in the Tutorial Component
In the following sections we describe the Tutorial Component in detail. We start by motivating the basic declarative approach used for the representation of the pedagogical knowledge. Then, Section 2.3 describes an ontology we developed that provides a vocabulary to specify the instructional purpose of learning materials, a necessary information for course generation. The learners’ possibilities of accessing the pedagogical knowledge of the Course Generator are described in Section 2.5. The subsequent sections walk through the sub-components of the Tutorial Component: first the Course Generator (Section 2.4), followed by the Exercise Sequencer (Section 2.6), the Suggestion Agent (Section 2.7), and the Tutorial Control (Section 2.8).

2.2 A Declarative, Task-Based Representation of Pedagogical Knowledge

An principal design decision regarding how to model human expert knowledge concerns the choice between a procedural or declarative representation. A procedural approach can be implemented directly in form of (Java) objects and methods. However, for modeling human knowledge, the procedural approach is hardly the appropriate level of abstraction. Therefore, we opted for a declarative representation. A declarative representation of knowledge eases authoring and change of that knowledge; and the area of Artificial Intelligence offers many frameworks to apply such declarative knowledge.

We also decided to distinguish between task and methods, i.e., between what to achieve and how to achieve it [81]. This clearly separates different kinds of knowledge, and, as we will see, provides an abstract layer that can be used for communication between the components of LEActiveMath. Additionally, using partly different collections of tasks and methods, various problem solving strategies can be realized.

2.2.1 Pedagogical Tasks

A pedagogical task corresponds to the goal a learner wants to achieve. It combines the two dimensions domain (content) and pedagogical objective. More specifically, a task \( t \) is defined as a tuple \( t = (l, C) \), where \( l \) is an identifier of a pedagogical objective and \( C \) is a set of identifiers of learning objects (content goals). While \( C \) specifies the course’s primary target concepts, the pedagogical objective determines the kinds of learning objects selected for \( C \).

The identifiers of the pedagogical objectives are taken from a set of terms that was derived from the pedagogical scenarios described in Deliverable D20. To begin with, each scenario corresponds to a pedagogical objective: LearnNew, Rehearse, GetOverview, TrainCompetency, Workbook and ExamSimulation. Additional pedagogical objectives help to achieve these scenario objectives. They include explain!, illustrateWithSingleExample!, trainWithSingleExercise! and motivate!.

As the Tutorial Component’s principal function is to support the learner in achieving learning goals, pedagogical tasks offer the appropriate level of abstraction to describe this functionality. Hence, the communication between within the Tutorial Component and between the Tutorial Component and other components will mainly consist of pedagogical tasks and learning materials selected for a task. Section 3.1.4 contains a detailed list of tasks and provides additional details on the communication.

\footnote{For now, the semantic of the exclamation mark is irrelevant. It will be explained in Section 2.4.1.}
2.2.2 Pedagogical Methods

Pedagogical methods describe how to achieve tasks. In the LeActiveMath system, this knowledge is encapsulated within a single component, the Course Generator. This means that, for instance, the selection of examples or exercises appropriate to the current learning situation is performed exclusively by the Course Generator. Other components that in some situations require such a functionality (such as the Open Learner Model and the assembly tool) transmit this task to the Course Generator (via the Tutorial Control) and receive appropriate content as a result. This way, we can ensure that a system-wide coherent pedagogical approach is followed throughout LeActiveMath. This avoids situations where one component reacts differently than a second one, potentially leading to confusion of the learner.

To summarize, pedagogical tasks and methods provide a vocabulary that describes the learning goals within the LeActiveMath system and how to achieve them. An additional vocabulary is required that describes the learning material from an instructional point of view. This will be motivated and explained in the following section.

2.3 Ontology of Instructional Objects

The pedagogical strategies of D20 describe what learning objects to select at specific times during the learning process. Consequently, the learning objects must be described in a way that makes this selection process possible. However, existing standards for describing learning object metadata fail to represent crucial information relevant from this pedagogical point of view.

The most widely accepted standard currently is the IEEE Learning Object Metadata [24]. LOM’s educational category partly allows a description of resources from an pedagogical perspective, in particular its learning-resource-type (property 5.3): its possible values are Exercise, Simulation, Questionnaire, Diagram, Figure, Graph, Index, Slide, Table, Narrative Text, Exam, Experiment, ProblemStatement, SelfAssessment. The problem with the values is that they mix pedagogical and technical information. While Diagram, Figure, Graph, Slide and Table describe the format of a resource, other values such as Exercise, Simulation and Experiment cover the instructional type. They represent different dimensions, hence need to be separated. Furthermore, several instructional objects are not covered by LOM (e.g., definition, example). Similar arguments apply to alternative vocabularies of the LOM learning-resource-type like GEM [21].

Other relevant e-learning standards in this context are IMS Learning Design (IMS LD, [29]) and IMS Question and Test Interoperability (IMS QTI, [27]). IMS LD describes ordered activities in learning and the roles of the involved parties. It does not represent single learning resources and their instructional functions. IMS QTI provides a representation of exercises which encodes common exercise types, such as multiple choice, image hot-spot, and fill-in-blank. It specifies the functionality of an exercise, not its pedagogical purpose.

Therefore, we designed an ontology of instructional objects that describes learning resources from an instructional perspective. The ontology does not describe the content taught by the learning material, e.g., concepts in mathematics and their relationships. Instead, each of its classes stands for a particular instructional role a virtual or text-book learning resource, for instance a paragraph in a text-book, can play.

The goal of this work was to derive a pedagogically sound ontology of types of learning resources that allows for efficient reuse. The principal design goals were:

**Domain independence** The types should be independent of the domain that is being taught, i.e., they should characterize learning material about mathematics as well as literature.
Pedagogical flexibility The types should be independent of the pedagogical principles underlying the learning material, i.e., they should describe LeActiveMath moderate constructivism as well as more traditional didactic approaches.

Completeness The types should cover the existing range of learning materials as much as possible.

Compatibility The mapping of existing standards and e-learning knowledge representations should be as easy as possible.

Applicability Authors should be able to understand and apply the ontology (LOM, for instance, is notorious for putting a heavy load on content developers).

Efficiency in searching Users should be able to narrow down the search for learning objects quickly using terms that reflect what they need.

Machine processability The types (together with other metadata) should enable sophisticated machine applications to find learning objects without human intervention.

In order to derive an ontology that complies to these goals as much as possible, it was necessary to analyze a significant amount of sources. Here, sources ranged from text classification ([53]), over instructional design (e.g., [70, 71, 20, 84, 16, 91, 57]) to knowledge representations of structured texts [6, 89, 63] or implemented in e-learning systems (e.g., [54, 35, 78, 52, 10, 29]). The ontology was evaluated and adapted iteratively. The last evaluation was done within the LeActiveMath project, among others by the e-learning lab of Klett. See Section 4.1 for details.

It is important to note that this ontology does not describe the content taught by the learning material, e.g., “definition of the derivative function”. Instead, each class of the ontology stands for a particular instructional role a learning object can play, for instance “An EXAMPLE for the definition of the derivative function” or “An INTRODUCTION for the definition of the derivative function”.

We decided to model the instructional type using an ontology for the following reasons. An ontology provides more information than a taxonomy (a hierarchy of terms). It allows to represent different relationships between the terms and how they are composed of different terms. For instance, using the ontology of instructional objects, an author can specify that a learning object is an example for another learning object. Compared to LOM’s resource-type which is a flat list, even a taxonomy has definite advantages. For instance, using the term hierarchy, a search engine could use the tree structure to widen the search and include siblings or parent nodes in case that for a specific query no element was found.

In the following, we will describe the classes and properties of the ontology of instructional objects, shown in Figure 2.4. The ontology was implemented using Protegé ([22]) and is available as an OWL file at http://www.ags.uni-sb.de/~cullrich/oio/InstructionalObjects.owl

Instructional Object. “Instructional Object” is the root node of the ontology. Several properties are defined at this level: a unique identifier; “learning context”, which describes the educational context of the typical target audience; “field”, which describes the field of the target audience. Additional properties correspond to the metadata defined in D6 (Structure and Metadata Model). The property “requires” is used to represent dependencies between instructional objects.

Fundamental A learning object of the type “Fundamental” conveys the central pieces of information about a domain that the learner should learn in an instructional process. Pure fundamentals are seldom found in learning materials. Most of the time, they come in the form of one of their specializations. Albeit fundamentals are not necessarily instruction-specific because they cover types of knowledge in general, they are included in the ontology because they are necessary for instruction: learning objects often have the instructional function of presenting a fundamental.
Fact A learning object of the type “Fact” provides information based on real occurrences; it describes an event or something that holds without being a general rule.

Definition A learning object of the type “Definition” states the meaning of a word, phrase, or symbol. Often, it describes a set of conditions or circumstances that an entity must fulfill in order to count as an instance of a class.

Law A learning object of the type “Law” describes a general principle between phenomena or expressions that has been proven to hold, or is based on consistent experience.


Theorem A learning object of the type “Theorem” describes an idea that has been demonstrated to be true. In mathematics, it describes a statement which can be proven true on the basis of explicit assumptions.

Process “Process” and its subclasses describe a sequence of events. The deeper in the class hierarchy, the more formal and specialized they become. A learning object of the type “Process” provides information on a flow of events that describes how something works and can involve several actors.

Policy A learning object of the type “Policy” describes a fixed or predetermined policy or mode of action. One principal actor can employ it as an informal direction for tasks, or a guideline.

Procedure A learning object of the type “Procedure” consists of a specified sequence of steps or formal instructions to achieve an end. It can be as formal as an algorithm.
**Auxiliary** A learning object of the type “Auxiliary” provides information about fundamentals that is not necessary for understanding the domain, but supports the instructional process. They motivate the learner, and offer engaging and challenging learning opportunities. Every auxiliary object offers information about one or several fundamentals. The identifiers of these fundamentals are enumerated in the property “isFor”.

**Interactivity** A learning object of the type “Interactivity” offers some kind of interactive aspect. It is more general than an exercise as it not necessarily has a defined goal that the learner has to achieve. It is designed to develop or train a skill or ability related to a fundamental. The subclasses of “Interactivity” do not capture technical aspects. In general, the way how an interactivity is realized, for instance as a multiple choice question, is independent of its instructional function. A well-designed multiple choice question can target knowledge as well as application of a fundamental.

**Exploration** Using a learning object of the type “Exploration”, the user can freely explore aspects of a concept without a specified goal, or with a goal but no predefined solution path.

**Real World Problem** A learning object of the type “RealWorldProblem” describes a situation from the learner’s daily private or professional life that involves open questions or problems.

**Invitation** A learning object of the type “Invitation” is a request to the learner to perform a meta-cognitive activity, for instance by using a tool.

**Exercise** A learning object of the type “Exercise” is an interactive element that requires the learner’s feedback. The feedback can be evaluated (either automatically or manually) and an success ratio can be assigned to it.

**Illustration** Learning objects of the type “Illustration” illustrate a concept or parts of a concept.

**CounterExample** A learning object of the type “CounterExample” is an instructional object that is not an example of a fundamental but is often mistakingly thought of as one.

**Example** Learning objects of the type “Example” positively illustrate the fundamental or parts of a fundamental.

**Evidence** A learning object of the type “Evidence” provides supporting claims made for a law or one of its subclasses. The “isFor”-property of an evidence has as range the class “Law”.

**Demonstration** A learning object of the type “Demonstration” represents a situation in which is shown that a specific law holds, e.g., experiments in physics or chemistry.

**Proof** A learning object of the type “Proof” is strict evidence consisting of a test or a formal derivation of a law.

**Explanation** A learning object of the type “Explanation” provides additional information about a fundamental. It elaborates certain aspect or points out important properties.

**Conclusion** A learning object of the type “Conclusion” sums up the main points of a fundamental.

**Introduction** A learning object of the type “Introduction” contains information that leads the way to a fundamental.

**Remark** A learning object of the type “Remark” provides additional, not mandatory information about an aspect of a fundamental. It can contain interesting side information, or details on how the fundamental is related to other fundamentals.

A representation of pedagogical knowledge that uses the terms defined in the ontology is independent of any concrete, system-specific knowledge representation. This has two advantages: first, the pedagogical knowledge can be re-used, i.e., applied in settings different from LEActiveMath, and second, external repositories can be integrated quite easily. With regard to the pedagogical strategies defined in this paper, the ontology provides a vocabulary used to describe the learning
objects that are selected during the execution of the strategies. Section 2.4.2, the description of a mediator component, will provide additional details and examples about these claims. The evaluations of the ontology are described in Section 4.1.

The ontology was recently used for a revised version of the ALOCoM ontology, a recent EU ProLearn NoE effort [34], in the e-learning platform e-aula [74], and in the CampusContent project of the Distant University Hagen [36].

2.4 Course Generator

The Course Generator encapsulates the pedagogical knowledge of selecting appropriate learning objects for a pedagogical task. It receives a pedagogical task as an input and returns a sequence of learning object identifiers as a result.

First, we introduce the planning framework chosen for the implementation of the pedagogical strategies. We then describe the mediator, a component that connects the planner with the learning object repositories. Section 2.4.3 then presents the implementation of the scenario “Learn-New”.

2.4.1 Hierarchical Task Network Planning

Applying Artificial Intelligence frameworks to real-life applications often raises unexpected problems. In this section, we describe the framework we used for the implementation of the pedagogical strategies, present challenges that we encountered and provide solutions for these challenges. Part of this work was done jointly with the Automated Planning Group of the University of Maryland, one of the leading AI planning groups.

2.4.1.1 Problem Solving in HTN-Planning

Hierarchical Task Network (HTN) planning is an AI planning framework that is based on hierarchical decomposition. In HTN planning, the goal of the planner is to achieve a partially or fully ordered list of top tasks, where each task is a symbolic representation of an activity to be performed. We will use the term HTN-tasks to distinguish these tasks from the pedagogical tasks described in Section 2.2.1. The planner formulates a plan by decomposing HTN top tasks into smaller and smaller subtasks until primitive HTN-tasks are reached that can be carried out directly. The basic idea was developed in the mid-70s [73, 83]. The development of the formal underpinnings came much later, in the mid-90s [17]. HTN-planning research has been much more application-oriented than most other AI-planning research, and most HTN planning systems have been used in one or more application domains [92, 12, 64].

A HTN planning problem consists of the following: the initial state (represented as a set of logical atoms that are assumed to be true at the time that the plan executor will begin executing its plan), the initial task network (a set of HTN-tasks to be performed, along with some constraints that must be satisfied), and a domain description that contains the following:

- A set of planning operators that describe various kinds of actions that the plan executor can perform directly. These are similar to classical planning operators such as the ones in PDDL [56, 18], with preconditions, add and delete lists. Each operator instance can carry out a primitive HTN-task associated with it. These operator instances change the world state upon their execution according to their add and delete lists.
• A set of methods that describe various possible ways of decomposing non-primitive HTN-tasks into subtasks. These are the “standard operating procedures” that one would normally use to perform tasks in the domain. Each method may have a set of constraints that must be satisfied in order to be applicable.

• Optionally, various other information such as definitions of external functions (i.e., code calls to external agents that the planner can make while evaluating a condition or calculating a binding during planning) and definitions of axioms (i.e., horn-clause like statements for inferring conditions that are not mentioned explicitly in world states, but can be proven using a theorem prover).

Planning is done by applying methods to non-primitive HTN-tasks to decompose them into subtasks, and applying operators to primitive HTN-tasks to produce actions. If this is done in such a way that all of the constraints are satisfied, then the planner has found a solution plan; otherwise the planner will need to backtrack and try other methods and actions.

The planner we used in LEActiveMath is JSHOP2 [25]. JSHOP2 is a compiler that takes a HTN domain description as input and compiles it into a set of domain-specific Java classes that can later be used to solve planning problems in that domain. These classes implement a domain-specific instance of a domain-independent planner. The fact that JSHOP2 is a compiler rather than an interpreter helps it in optimizing the domain-dependent code it produces [26].

Figure 2.5 contains an example of an HTN-operator in JSHOP2 syntax. There, the semicolon indicates the start of a comment, terms starting with a question mark stand for variables, primitive HTN-tasks are marked with an exclamation mark, and a double exclamation mark denotes task only relevant for internal bookkeeping.

The operator in the example is applicable given a) that the HTN-task (!a ?c) can be matched against a not yet achieved HTN-task and b) that the precondition p holds (i.e., an atom exists in the world state that can be matched with it). In case an operator is applied all atoms contained in the delete list are removed and all atoms contained in the add list are added to the world state respectively. As an example, if the world state contains p and d, then after the application of the operator the world state will contain p and a.

Figure 2.6 contains an example of a HTN-method. The method is applicable in case an open HTN-task exists that matches with (t ?c) and any of the precondition lists holds. The preconditions are tried in the given order. If one matches, the HTN-task are decomposed in the corresponding subtasks (in the example, t1, t2 and !a for the preconditions p, t3 for the preconditions p2).

During the implementation of the pedagogical strategies, we encountered several problem which were due to the real-life application of the framework in the LEActiveMath e-learning system. In a joint work with the developer of JSHOP2 from the planning group of University of Maryland we identified these challenges and put them in the broader context of distributed information systems. In the following section, we describe the challenges and propose our solutions.

2.4.1.2 Challenges of HTN Planning in Distributed Information Systems

Distributed information systems integrate data from a set of different, heterogeneous sources scattered all over the Web. Figure 2.7 provides a schematic overview of an information system.
Figure 2.6: A HTN-method

Figure 2.7: The different layers of an information system

as defined in [3]. According to this definition, the client accesses the system via the Presentation Layer, which allows the client to interact with the system and renders the information. The data processing is done in the second layer, the Application Logic Layer. This layer accesses information or resources offered by the third layer, the Resource Management Layer. In case of a distributed information system, the resources reside in several, potentially independent repositories.

Especially in complex settings, the application logic of a distributed information system can make use of knowledge-intensive problem solving. In such cases AI planning frameworks such as HTN-planning can prove very useful. However, HTN-planning comes with several inherent properties that make it difficult to apply in a straightforward way. This section discusses these difficulties in detail.

Vast Amounts of Resources Often, a distributed information system needs to access a vast amount of resources. Difficulties arise in case the resources (or descriptions of them, i.e., metadata) serve as a basis for the reasoning process of the planner. For instance, during course generation in a Web-based learning environment such as LEACTIVEMATH, decisions need to be taken depending on whether a specific learning object exists that fulfills given criteria. As an example, consider the case that an introduction to the mathematical concept “Definition of Derivation” needs to be generated. If no textual introduction to this concept exists as a learning object, then an alternative is to use a learning object that is an easy example for the concept.

Moreover, only a limited number of learning objects will be required for planning a specific course, typically restricted to the part of the subject domain the learner is currently working on.
These examples illustrate that the reasoning process takes into consideration information about available resources. However, this involves the difficulty that traditional AI planning requires evaluating a method’s precondition against the planner’s world state. In a naive approach, this would require mirroring in the world state all the information available about the resources in all the repositories. In real world applications, this is simply infeasible. Additionally, only a subset of all stored resources may be relevant for the planning, but it is impossible to know which subset this is in advance.

**Distributed and Heterogeneous Resources** Additional difficulties can arise in case the resources are distributed over several repositories, and the resource management is not restricted within a single component but divided into several components, and each is accessed by the application logic separately. In this case, often the representation of the resource varies simply because different database schemas may be used. Unfortunately, in the e-learning example such heterogeneous resources are the norm. Despite standardization efforts such as LOM, almost every repository available uses its own description of learning objects or at least a variant of LOM. But still, the course generation planning needs to evaluate constraints about the existing learning objects, in the same way as described above, regardless of how they are described. One wants to avoid different operators and methods for each possible repository. Generally speaking, the reasoning process needs to be able to handle different kinds of data which represent the same information in the same way (i.e., by using the same operators and methods), regardless of the origin and actual representation of the data.

**Changing vs. Stable Information** In addition to the problems described above, some information required during the planning process may change rather frequently while some remains stable over a longer period of time. For optimization issues, it proves efficient to handle these two kinds of resources differently. As described above, a typical example of frequently changing information in a Web-based learning environment is information about the learner’s knowledge levels. These values (hopefully) change after each interaction with a learning object. On the other hand, the information about the learning objects themselves only will change in case content is added, deleted, or modified.

**Third-Party Services** Often, the flow of execution in the application logic layer depends on other services. During the planning process, the planner may need to access the information provided by a service or at least need to know whether or not a service is available. However, one wants to avoid using a different domain description for each potential configuration. Generally speaking, the planning process needs to be able to access information provided by the services as well as information about their availability, since access to these services may vary, depending on the configuration of the overall system. Therefore, the planning needs to take into account dynamic information about these services.

Solutions to these challenges are explained in the following sections. We will start with describing a general architecture used for integrating distributed and heterogeneous resources.

### 2.4.2 A Mediator for Repository Integration

The mediator described in this section allows LEACTIVEMATHE's Course Generator to access information from multiple repositories and not exclusively those of LEACTIVEMATHE. For that purpose we introduce a mediating architecture that translates queries for a specified set of connected repositories and passes the translated queries to the repositories. The advantage of a mediating architecture is that the querying component, e.g., the Course Generator, does not need to know the specification of the data sources and their query languages [90]. An ontology-based query-rewriting mechanism integrated in our architecture enables the integration of new repositories.
The mechanism uses the specified knowledge representation of the repository to be integrated and an ontology mapping to compute the rewriting steps for translating queries sent to the new repository.

2.4.2.1 Queries and Query Language

The mediator provides a single interface for querying several data sources. This interface accepts a query language that specifies metadata of learning objects; therefore, a query sent to the mediator contains a metadata specification of learning objects and returns the Uniform Resource Identifiers (URIs) of the learning objects which meet this specification. A query is comprised of three parts:

- **RelationQueries** comprise the relational metadata the learning objects to be retrieved has to meet. It is a set of triples \((relation, relation, LO)\) in which the keyword \(relation\) denotes the type of the query part, \(relation\) specifies the relation between the learning object \(LO\) and the learning objects to be retrieved.

- **ClassQueries** comprises all classes the learning objects to be retrieved belong to. It is a set of pairs \((class, class)\) in which \(class\) denotes the category the returned learning objects belong to.

- **PropertyQueries** comprise the metadata of properties. It consists of a set of triples \((property, property, value)\). Each queried learning object satisfies each property-value-pair \((property, value)\).

A query asking for all learning objects which are easy exercises training the concept \textit{asymptote} looks as follows:

\[(relation isFor asymptote)(class Exercise)(property hasDifficulty easy)\].

2.4.2.2 Ontology Mapping and Query Rewriting

Queries sent to the mediator contain terms defined in the ontology of instructional objects (OIO) described in Section 2.3. These terms have to be replaced with the corresponding terms a repository uses to describe its learning objects.

Consider a repository using the term \textit{trains} for expressing the relation between an exercise and its topic item and the boolean property \textit{isDifficult} to express whether a learning object is difficult or not. For that repository the aforementioned query has to be translated into the query

\[(relation exercises asymptote)(property isDifficult true)\]

To guarantee the correct substitution of the terms appearing in the queries, one has to define an ontology mapping between the OIO (as the source ontology) and each of the pedagogical ontologies describing the metadata structure of the target repositories. Therefore, to define an accurate mapping, it is necessary to make explicit and specify an ontology representing the knowledge structure and metadata semantics of each target repository.

An XML-based ontology mapping language represents the mappings between the OIO and the target ontologies. Generally, an ontology mapping expresses the semantic overlap between two ontologies \(O_S\) and \(O_T\) [13]. An ontology mapping can be one-way or two-way [67]. A one-way
An ontology mapping specifies how to express the metadata of a concept (formulated in terms of the source ontology $O_S$) in terms of the target ontology $O_T$. It expresses which concepts of $O_S$ are semantically contained in which concepts of $O_T$. Two-way mapping works both ways, hence they express semantical equivalence between concepts. For our rewriting approach a one-way mapping is sufficient since we are exclusively interested in mappings from the OIO into the target ontologies.

An ontology mapping contains a set of mapping patterns (see Figure 2.8 as an example). Each mapping pattern consists of a matching pattern and a set of replacement patterns. Both replacement and matching patterns specify a concept by restricting the category a concept belongs to (ClassRestriction-Element), the property a concept has (PropertyRestriction-Element), and/or the relation a concept connects with other concepts (RelationRestriction-Element). The mapping pattern expresses the semantical containment between the concept specified in the matching pattern and the concepts specified in the corresponding replacement patterns.

We say a mapping pattern $M$ matches a query $Q$, if $Q$ contains each term specified in the matching pattern of $M$. Applying $M$ to the query $Q$ means each term appearing in the matching pattern of $M$ and in $Q$ is deleted from the query and replaced by the terms specified in the replacement patterns belonging to $M$. Hence, applying a mapping pattern containing $n$ replacement patterns implies the creation of $n$ new queries since a mapping is possibly not unique.

**Pattern Overlapping** Overlapping of mapping patterns (see the second and third mapping pattern in Figure 2.8) are managed by a partial order on matching patterns which defines that one mapping pattern is more specific than another one. The mapping procedure guarantees that the most specific pattern is applied. For an example, consider the query

```xml
<OIOMapping>
  ...
  <MappingPattern>
    <MatchPattern>
      <ClassRestriction name="Introduction"/>
    </MatchPattern>
    <ReplacementPattern>
      <ClassRestriction name="context"/>
      <PropertyRestriction name="type" expected_value="introduction"/>
    </ReplacementPattern>
    <ReplacementPattern>
      <ClassRestriction name="context"/>
      <PropertyRestriction name="type" expected_value="motivation"/>
    </ReplacementPattern>
  </MappingPattern>
  <MappingPattern>
    <MatchPattern>
      <ClassRestriction name="Example"/>
      <RelationRestriction name="isFor"/>
    </MatchPattern>
    <ReplacementPattern>
      <RelationRestriction name="example_for"/>
    </ReplacementPattern>
  </MappingPattern>
  <MappingPattern>
    <MatchPattern>
      <RelationRestriction name="isFor"/>
    </MatchPattern>
    <ReplacementPattern>
      <RelationRestriction name="for"/>
    </ReplacementPattern>
  </MappingPattern>
  ...
</OIOMapping>
```

Figure 2.8: Excerpt of a mapping
Figure 2.9: Mediator Architecture

and the ontology mapping shown in Figure 2.8. Although both mapping patterns match, one has to make explicit that the second mapping pattern is the better choice for the query than the third one is.

Applying an ontology mapping to a query is defined as applying the most specific mapping pattern that matches a query. A term for which no matching pattern is found is left as it is. The rewriting procedure considers this term as used in in both ontologies. This approach avoids that one has to write mapping patterns expressing the pure identity of terms.

Applying the ontology mapping of Figure 2.8 to the query

(class Example)(relation isFor asymptote)

yields the following two new queries (whose results are aggregated).

1. (class omtext)(relation for asymptote)(property type introduction)
2. (class omtext)(relation for asymptote)(property type motivation)

Query Expansion The expansion of a query guarantees that the mediator, if asked for category \( C \), returns not only objects belonging to \( C \) directly but also objects belonging to subcategories of \( C \). For a query \( Q \), the query expansion algorithm returns \( \prod_{i=0}^{n} m(C_i) \) queries, where \( C_0,...,C_n \) are the categories specified in \( Q \) and \( m(C_i) \) counts all recursive subcategories of a category \( C_i \).

2.4.2.3 Architecture

Our architecture is a mediation information system architecture as introduced by Wiederhold [90]. Its main component, called mediator, provides a uniform interface for accessing multiple heterogeneous data resources. It accepts queries formulated in the query language described above and returns a set of URIs, where each URI points to a learning object that meets the conditions specified in the query. For each repository, a wrapper object is integrated. It encapsulates the following information: the specification of the ontology of the repositories knowledge (as an OWL-ontology-definition) and the mapping between the terms of the ontology of instructional objects and the terms the repository uses (see Figure 2.9). The mediator utilizes the OWL file for query expansion and the mapping specification for query rewriting.
2.4.2.4 Wrappers

After rewriting, the queries have to be passed to the repositories. To manage the different querying technologies the mediator comprises a set of wrappers. Each wrapper queries the corresponding repository for metadata by creating repository specific commands. These commands are implemented in the following three methods that each wrapper offers to the mediator:

- `queryClass` returns the set of categories a given item belongs to.
- `queryProperty` returns the set of property-value pairs a given item has.
- `queryRelation` returns a set of URIs a given item is related to.

Let us stress that our approach focuses on mapping of concepts and not on mapping of instances of the subject domain. To provide an architecture which allows the generation of mixed courses, i.e., courses comprising learning objects from different repositories, we are planning to integrate an instance mapping technology basing on domain ontologies and mappings between them.

2.4.2.5 Caching

As stated in [77], a repository is not always equipped with a powerful caching mechanism. Hence it is reasonable to integrate a caching mechanism into the mediating component. If a query is sent a second time, the mediator does not query each connected repository again. It returns the cached set of URIs instead, which reduces run-time complexity dramatically. The results of parts of queries are cached, too. The cache is invalidated when the mediator is shutdown, i.e., when the LEActiveMath is stopped. We were able to confirm the performance increase in an evaluation (see Section 4.2.2 for details).

The validity of the mediator approach and of the ontology of instructional objects has been supported by the successful integration of repositories outside the LEActiveMath-project, namely the DaMiT-system [31] and the MathsThesaurus [85]. See Section 4.1 for details.

2.4.3 Representing Course Generation Knowledge in a HTN-planner

This section covers the implementation of the pedagogical strategies formulated in D20. We will use the scenario “LearnNew” as an example and show how its description can be expressed in the HTN framework. The complete domain description is contained in the separate available appendix. We did not include it in this part of the deliverable because it would have added about another hundred pages.

2.4.3.1 Implementation of the Scenario LearnNew

The descriptions of the pedagogical scenarios in D20 [47] are hierarchical. Take “LearnNew” (see Figure 2.10) as an example: “In order to provide a course about new content for a learner, one should start with an introduction, then describe the involved concepts, . . . ”. Each step can be described hierarchically, again: “For the introduction, first provide a motivation and a problem involving the concepts, then potentially provide some exercises, . . . “. We found that the formalization of the knowledge into planning operators and methods of HTN-framework was quite straight-forward.

The implemented pedagogical strategies consist of about 250 methods, 20 operators and 40 axioms. To limit the amount of pages in this deliverable, we will describe one scenario (“LearnNew”) in
sufficient detail to explain our approach. Please note that the implementation differs in some minor aspects from the formalization in D20 (see Section 5.2 for details).

In JSHOP2, a task is defined as a predicate symbol followed by several arguments. Hence, a pedagogical task can be mapped to a HTN-task by mapping the pedagogical objective onto the predicate symbol, and the content goals onto the arguments. Because of this straightforward mapping, we will no longer distinguish between pedagogical and HTN-tasks from now own.

The task \( \text{learnNew} \text{identifiers} \) serves as the top-level task that starts the course generation in the scenario “LearnNew”. Figure 2.11 illustrates the methods that handle this task. The top method decomposes it into four sub-tasks: \(!\text{startSection}\) and \(!\text{endSection}\) are primitive tasks, which serve to generate structure by opening and closing sections (chapters) within a course. The fourth sub-task \( \text{reflect ??concepts} \) inserts a section that provides meta-cognitive support by suggesting to reflect on the achieved learning progress. The usage of learning services to support such activities will be discussed later in this section. The second task, \( \text{learnConceptsLearnNew ??concepts} \), is processed by the two remaining methods of Figure 2.11: \( ??concepts \) is instantiated with a list of concept identifiers, the second method recursively walks through the list and inserts the sub-task \( \text{learnConceptLearnNew} \) for each identifier in the list. The bottom method catches the case for the empty list and stops the recursion.

As a result, for each concept \( c \) of the initially provided content goals, the task \( \text{learnConcept-LearnNew c} \) will be created. Figure 2.12 illustrates the method used to achieve this task. Its sub-task closely resembles the structure of the scenario “LearnNew” as illustrated in Figure 2.10.

In order to keep methods and tasks manageable, their naming follows an informal scheme, which is not imposed by the JSHOP2 syntax. First, non top-level tasks and methods that trigger the creation of a section have the suffix \text{Section}. Second, the suffix “!” is used to distinguish
between critical and optional tasks: by default, HTN-planning will fail as soon as a task can not be achieved. However, in the case of course generation, supporting the learner with a suboptimal course is preferable to providing no support at all. Course generation should be a best effort and should only fail in a very limited number of cases. As a consequence, we distinguish between critical and optional tasks. Critical tasks represent necessary elements of the pedagogical strategy that have to be fulfilled for the course generation not to fail. For instance, in a problem-based approach it is mandatory to present a learning object that is of the type \textit{Problem}. Critical tasks are marked with the suffix "!". In contrast, optional tasks should be achieved if possible, but failing to do so will still result in a course. As an example, the subtask \texttt{introduceWithPrereqSection} in Figure 2.12 is optional, e.g., if there are no learning object that can serve to introduce the concept \(?c\), the concept will still be considered as introduced.

Technically, the distinction between critical and optional tasks is realized by encapsulating critical tasks with \texttt{fallback} methods. As an example, Figure 2.13 shows how two methods are used to achieve the optional task \texttt{(introduceWithPrereqSection ?c)}. First, by the upper method, the critical task \texttt{(introduceWithPrereqSection! ?c)} is added. In case it can not be achieved, the second method is applied. As it contains no sub-tasks, it directly achieves the task \texttt{(introduceWithPrereqSection ?c)}.

Figure 2.14 shows two of several alternative methods that introduce a concept. Again, the structure induced by the task decomposition matches the structure as provided in Figure 2.10. The first method separates the presentation of an introduction from presenting the prerequisites of a concept. Such decomposition eases the re-usability of tasks; if a different scenario requires presenting prerequisites without an introduction, then the methods defined here can be reused.

In the second method, the sub-task \texttt{(motivate! ?c)} is critical; in other methods, not shown in...
the figure, different sub-tasks are critical.

The primitive sub-task (!text Introduction (?c)) triggers the creation of a template used for natural language generation. The parameters define the type and scope of the text. Section 3.2.7 provides additional technical details.

Figure 2.15 contains a subset of the methods used to achieve the motivation of a concept. Intuitively, the first method is applicable in case the learner exhibits no math anxiety and there is a learning object that is an exercise as well as an introduction for the concept ?c. If these preconditions do not hold, the second method checks whether there exists a learning object that is an example as well as an introduction. In case there is no such learning object, the third method is tried: is there a learning object that is an introduction for the concept? These methods all require that the learning context of the retrieved learning object is equal to the educational level of the learner. This condition is relaxed in the fourth method, which checks whether there is an introduction of a non-optimal learning context.

These methods illustrate an important technical aspect of the course generation process, namely access to information in the resource layer, in this case information about the learner and learning objects. In Section 2.4.1.2 we argued that access to resources has to be managed intelligently; mirroring all resources in the world state of the planner is practically impossible. The solution we propose is based on external functions. An external function as described in Section 2.4.1 serves to calculate information not directly available in the world state using procedures not native to the planning algorithm. Thus, instead of the operator's (or method's) preconditions to match against the logical atoms that make up the world state, they invoke function calls that return possible substitutions.

As an example, first consider the standard case where all information is represented in the world state. Let's assume that the two atoms (a 1) and (a 2) hold. An operator whose precondition contains (a ?n) will generate the substitutions ?n → 1 and ?n → 2.

In case the information about a is not represented in the world state but in a repository, an external function needs to be used. This results in a precondition similar to (assign ?n (call
(:method (motivate! ?c)
  (;; preconditions:
   (learnerProperty hasEducationalLevel ?el)
   (learnerProperty hasAnxiety ?c ?an)
   (call <= ?an 2)
   (assignIterator ?element
    (call GetElements (;; the constraints:
      (class Exercise)
      (class Introduction)
      (relation isFor ?c)
      (property hasLearningContext ?el)
      (property hasDifficulty very_easy)
    )))
  ;; sub-task:
  ((insertIfReady! ?element)))
)

(:method (motivate! ?c)
  ((learnerProperty hasEducationalLevel ?el)
   (assignIterator ?element
    (call GetElements ((class Example)
      (class Introduction)
      (relation isFor ?c)
      (property hasLearningContext ?el)
      (property hasDifficulty very_easy))
    )))
  ((insertIfReady! ?element)))
)

(:method (motivate! ?c)
  ((learnerProperty hasEducationalLevel ?el)
   (assignIterator ?element
    (call GetElements ((class Introduction)
      (relation isFor ?c)
      (property hasLearningContext ?el)
    )))
  ((insertIfReady! ?element)))
)

(:method (motivate! ?c)
  ((learnerProperty hasAllowedEducationalLevel ?aels)
   (assignIterator ?el ?aels)
   (assignIterator ?element
    (call GetElements ((class Introduction)
      (relation isFor ?c)
      (property hasLearningContext ?el)
    )))
  ((insertIfReady! ?element)))
)

Figure 2.15: Methods for motivating a concept
GetInfoOnA)). The assign command is used by JSHOP2 to bind a variable to a given value. call invokes the specified external function with the specified parameters (if any).

In the above example, \(?n\) is bound to the result of the external function. Whatever happens in the external function is hidden from the planner. The call may for instance involve queries to the resource layer.

However, this approach fails if there is more than a single possible value for the variable. In this case, all possible substitutions need to be generated. This is achieved by the axioms shown in Figure 2.16. The dot operator in JSHOP2 has the same meaning as in the LISP programming language, i.e., it separates the first element of a list from the rest of it. Using these axioms, if the precondition of an operator or method contains \((\text{assignIterator } ?n \ (\text{call GetInfoOnA}))\) and the call function returns a list of all possible values, subsequently all substitutions will be generated: The first axiom generates a substitution for the first value of the list; the second axiom recurses in the tail of the list. The third axiom is used (in this case) for binding a variable to a given value.

This approach is used in the fourth precondition of the methods in Figure 2.15. There, the predicate GetElements conditions sends a request to the mediator (see Section 2.4.2) to search the repositories for elements that fulfill the given conditions. The returned list of identifiers (if any) are subsequently bound to the variable ?element. The subtask (insertIfReady! ?element) inserts the element in the course if it was not already inserted earlier and either the learner has the necessary competence to understand the concept or it is explained in the course. Otherwise, the next binding of the variable ?element is tried. In case all bindings were tried unsuccessfully, the method is not applicable.

The methods in Figure 2.15 also access information about the learner. The first precondition of the top method \((\text{learnerProperty hasEducationalLevel } ?el)\) retrieves the educational level of the learner (e.g., secondary education or university first year) and binds it to the variable ?el. The second and third precondition check whether the math anxiety of the learner regarding the current concept is small (less than two). Figure 2.17 shows one of the axioms used to retrieve learner information. Again, an external function is used.\(^5\)

Coming back to the description of realizing the introduction of a concept (Figure 2.14), the tasks (problem ?c) and (introductionExemplify ?c) insert in the course a learning object of the type problem and an easy example that illustrates the application of the concept.

\(^{5}\)They keyword :first is used in JSHOP2 to control backtracking and indicates that only the first possible binding is applied (similar to the cut operator in Prolog).
In the final step of the introduction, the prerequisites are included in the course. The precondition of the method illustrated in Figure 2.18 first use an axiom to collect all prerequisites of the current concept. In case there are any, the method inserts sub-tasks that start a new section, trigger the generation of a text that explains the purpose of this section, and inserts all collected prerequisites.

Figure 2.19 contains the method encoding the knowledge how to develop a concept. All precondition-sub-task pairs insert a task that triggers the generation of a text template that explains the goal of the current section and the concept itself. They differ in the learner properties they take into account. The upper part of the method is applicable in case the learner knows the concept well, and inserts an example only. If the learner has low knowledge but is motivated, then a text explaining the concept is inserted, together with an easy exercise and several examples illustrating the concept. The bottom part of the method is applied in case the previous preconditions do not match, and inserts an explaining text along with examples.

The method illustrated in Figure 2.20 triggers the insertion of sub-tasks that ensures that a variety of exercises are presented to the learner. If the corresponding learning objects are available, then each competency is trained at each difficulty level.
(:method (train! ?c)
  MethodTrain!
  ()
  (trainWithSingleExercise ?c very_easy think)
  (trainWithSingleExercise ?c very_easy solve)
  (trainWithSingleExercise ?c very_easy represent)
  (trainWithSingleExercise ?c very_easy language)
  (trainWithSingleExercise ?c very_easy model)
  (trainWithSingleExercise ?c very_easy argue)
  (trainWithSingleExercise ?c very_easy tools)
  (trainWithSingleExercise ?c very_easy communicate)
  (trainWithSingleExercise ?c easy think)
  (trainWithSingleExercise ?c easy solve)
  (trainWithSingleExercise ?c easy represent)
  (trainWithSingleExercise ?c easy language)
  (trainWithSingleExercise ?c easy model)
  (trainWithSingleExercise ?c easy argue)
  (trainWithSingleExercise ?c easy tools)
  (trainWithSingleExercise ?c easy communicate)
  (trainWithSingleExercise ?c medium think)
  (trainWithSingleExercise ?c medium solve)
  (trainWithSingleExercise ?c medium represent)
  (trainWithSingleExercise ?c medium language)
  (trainWithSingleExercise ?c medium model)
  (trainWithSingleExercise ?c medium argue)
  (trainWithSingleExercise ?c medium tools)
  (trainWithSingleExercise ?c medium communicate)
  (trainWithSingleExercise ?c difficult think)
  (trainWithSingleExercise ?c difficult solve)
  (trainWithSingleExercise ?c difficult represent)
  (trainWithSingleExercise ?c difficult language)
  (trainWithSingleExercise ?c difficult model)
  (trainWithSingleExercise ?c difficult argue)
  (trainWithSingleExercise ?c difficult tools)
  (trainWithSingleExercise ?c difficult communicate)
  (trainWithSingleExercise ?c very_difficult think)
  (trainWithSingleExercise ?c very_difficult solve)
  (trainWithSingleExercise ?c very_difficult represent)
  (trainWithSingleExercise ?c very_difficult language)
  (trainWithSingleExercise ?c very_difficult model)
  (trainWithSingleExercise ?c very_difficult argue)
  (trainWithSingleExercise ?c very_difficult tools)
  (trainWithSingleExercise ?c very_difficult communicate)))

Figure 2.20: A method for selecting a variety of exercises
(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
MethodSelectExerciseHighMotivationDiffComp
((learnerProperty hasField ?field)
(learnerProperty hasEducationalLevel ?el)
(learnerProperty hasMotivation ?c ?m)
(caller = 4)
(learnerProperty hasCompetencyLevel ?c ?cl)
(assignIterator ?exercise
(call
GetElements
((class Exercise)
(relation isFor ?c)
(property hasLearningContext ?el)
(property hasCompetencyLevel (call + 1 ?cl))
(property hasField ?field)
(property hasDifficulty ?difficulty)
(property hasCompetency ?competency)
)))))
((insertWithVariantsIfReady! ?exercise ?c)))

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
MethodSelectExerciseAdequateCompetencyLevelDiffComp
((learnerProperty hasField ?field)
(learnerProperty hasEducationalLevel ?el)
(learnerProperty hasCompetencyLevel ?c ?cl)
(assignIterator ?exercise
(call
GetElements
((class Exercise)
(relation isFor ?c)
(property hasLearningContext ?el)
(property hasCompetencyLevel ?cl)
(property hasDifficulty ?difficulty)
(property hasField ?field)
(property hasCompetency ?competency)
)))
((insertWithVariantsIfReady! ?exercise ?c)))

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
MethodSelectExerciseAnyHasFieldDiffComp
((learnerProperty hasAllowedEducationalLevel ?els)
(assignIterator ?el ?els)
(learnerProperty hasCompetencyLevel ?c ?cl)
(assignIterator ?exercise
(call
GetElements
((class Exercise)
(relation isFor ?c)
(property hasLearningContext ?el)
(property hasCompetencyLevel ?cl)
(property hasDifficulty ?difficulty)
(property hasField ?field)
(property hasCompetency ?competency)
)))
((insertWithVariantsIfReady! ?exercise ?c)))

Figure 2.21: Selecting an exercise
The methods in Figure 2.21 are a subset of 13 methods that encode the pedagogical knowledge of selecting an exercise of a given difficulty level and competency. The upper method checks whether the user is highly motivated. Then, it searches for an exercise that would be slightly too difficult under normal circumstances (competency level plus 1). Additionally, the learning context of exercise should correspond to the educational level of the learner, and, similar, its field should correspond to the field of the learner.

The middle method covers the typical case and inserts an exercise with the competency level, learning context and field corresponding to those of the learner.

Best Effort Selection of Learning Objects In practice, it is impossible to encode course generation knowledge that covers each potential combination of information about the learner. Oversimplified, if a learner model represents \( n \) different types of information with \( m \) possible values each, then the number of possible combinations is \( n^m \). Writing rules or methods for each case would keep a lot of teachers busy for a long time.

Therefore, our methods encode the best possible as well as alternative ways of selecting learning objects. An ideal method typically takes a larger number of user properties into account and poses a larger number of constraints on the learning object.

In case no ideal method is applicable, fallback methods come into play. They encode the least constraining conditions on the learning objects possible and serve to insert elements in case no ideal method can be fulfilled. The bottom method of Figure 2.21 matches if any exercise for concept \( c \) exists that is of the correct competency level and has a learning context equal or lower as the educational level of the learner (provided by the predicate \( \text{hasAllowedEducationalLevel} \)).

In these methods, the insertion of an element is handled by the task \( \text{insertWithVariantsIfReady}! \). Often, auxiliary elements such as examples and exercises have variants, i.e., elements that describe the same situation but are formulated slightly differently, e.g., resulting in more difficult or easier exercises. For the exercise selection described here, the Course Generator needs to ensure that only a single variant is inserted in the course. The methods to achieve this task realize this by following the relation \( \text{variantOf} \) and adding atoms to the world state that mark these elements as inserted in the course. This way, the elements will not be selected later on. Also, they will not appear in the generated course, because the operator \( !\text{insert} \) was not applied.

The methods used for example selection (not shown here) roughly correspond to those for exercises. The major difference lies in the ordering. Examples serve to increase motivation by presenting the exemplary application is taken from the field of interest of the learner, even in case its assigned competence level is higher than the current competence level of the learner.

The methods responsible for the realization of the step connections & transfer are shown in Figure 2.22. This step serves to display theorems connected to the current concept. The upper method checks whether a concept mapping tool is available, and if so uses it (see the next paragraph for learning service integration). Otherwise, it triggers the display of theorems together with either proofs or proof exercises, depending on the competencies of the learner (bottom method).

Service Methods The final step in the scenario “LearnNew” consists of the reflection phase. Reflection is supported using a learning service, the Open Learner Model. The integration happens by \( \text{service-adding} \) methods. These methods encode the pedagogical knowledge at what time in the course the learner should use a tool and insert calls to the tool at the appropriate place in the course structure. Later, when the learner navigates through the course, an invitation to use the service is presented to her. Technical details on the generic integration of services are provided in Section 3.1.6.
(:method (connect! ?c)
  MethodConnectCMap!
  ((learningServiceAvailable CMap))
  ((connectByCMap! ?c))
  MethodConnectShowTheorems!
  ()
  ((connectByTheoremWithProof! ?c)))

(:method (proof! ?theorem)
  MethodProofHighArgueComp!
  ((learnerProperty hasCompetencyArgue ?theorem ?argue)
    (call >= ?argue 3)
    (learnerProperty hasEducationalLevel ?el)
    (assignIterator ?exercise
      (call GetElements (
        (class Exercise)
        (relation isFor ?theorem)
        (property hasLearningContext ?el)
        (property hasCompetencyArgue ?argue)
      )))
    ((insertIfReady! ?exercise ?theorem)))

  MethodProofLowArgueComp!
  ((learnerProperty hasCompetencyArgue ?theorem ?argue)
    (call < ?argue 3)
    (learnerProperty hasEducationalLevel ?el)
    (assignIterator ?proof
      (call GetElements (
        (class Proof)
        (relation isFor ?theorem)
        (property hasLearningContext ?el)
      )))
    ((insertIfReady! ?proof ?theorem)))

Figure 2.22: A selection of the methods implementing the step connections

(:method (reflect ?concepts)
  MethodReflectWithOLM
  ((learningServiceAvailable OLM))
  ((startSection Reflection)
    ((insertLearningService OLM display ?concepts (competencyId competency))
      (endSection))
  )

  MethodReflectManual
  ()
  ((startSection Reflection)
    (text Reflect ?concepts)
    (endSection)))

Figure 2.23: Using an learning supporting service for reflection
The checking for the availability of a service is done by introducing an external function, too. Thereby, a method’s preconditions can test for the availability of a service. In case the service is available, its call is inserted into the course. Otherwise, an alternative method is applied. This way, the pedagogical knowledge remains reusable, regardless of the actual configuration of the learning environment.

Figure 2.23 contains the methods used for integration the OLM. The upper method employs the axiom learningServiceAvailable to check whether the tool is available in the current configuration of the learning environment. If so, a call to the learning service is inserted in the course. Otherwise, the reflection is simply triggered by showing a text to the learner via the task (!text Reflect ?concepts). Section 3.2.5 describes the integration of the OLM in detail.

This concludes the description of the implementation of the scenario “LearnNew”. In the following sections, we will describe some novel features that this framework allows.

2.4.3.2 Providing Structure and Adaptivity: Dynamic Tasks

Course generation faces the following dilemma: on the one hand it should generate a structure that supports the learning process from the start on. On the other hand the selection of the learning objects should take into account the most up-to-date information about the learner that is available. However, if a long time-span lies between the generation and viewing of a page, some assumptions taken during course generation may have become invalid, resulting in an inadequate course.

Therefore, we extended the planner in such a way that planning may stop at the level of specially marked tasks (dynamic tasks). These tasks are inserted into the course like any other reference to a learning object. However, when the learner first visits a page that contains a dynamic task, this task is passed on to the course generator. The resulting identifiers of learning objects replace the task in the course structure with specific instances of learning objects (hence, when the page is revisited, the elements do not change, which avoids confusion of the learner reported in [4]). This means a course is partly static, partly dynamic.

Dynamic task execution offers new possibilities for authors, too. An author can define a course structure where parts of her course are predefined and others dynamically computed, taking the learner model into account. In this way, an author can profit from the best of both worlds: she can compose parts of the course by hand and at the same time profit from the adaptive features of the course generator.

Although the course generator supports dynamic tasks, the currently implemented scenarios do not use this feature yet. We preferred to complete the implementation of the scenarios before analyzing which tasks are best made dynamic. We will then revise the scenarios.

2.4.3.3 Converting a Plan into a Table of Contents

The result of the course generation is a plan, a sequence of operators as shown in Figure 2.24. From this plan, a table of contents represented in OMDoc, the data structure used in LEActiveMath for representing the content, is constructed the following way:

- !startSection triggers the opening of OMDoc’s grouping element (omgroup) and uses templates for generating the title of the corresponding section.
- !endSection insert the closing tag of an omgroup element.
- !insertElement inserts the ref element that OMDoc uses for referencing to OMDoc elements.
Figure 2.24: Parts of a plan generated by the course generator
The resulting OMDoc grouping therefore consists of nested sections with the leaves being pointers to learning objects. There exist several e-learning standards that represent similar structures, the most prominent being IMS Content Packaging (IMS CP, [28]), IMS Learning Design (IMS LD, [29]), and IMS Simple Sequencing (IMS SS, [30]).

In Section 1.3.3 we argued that IMS SS with its explicit control of the navigation process does not adhere to the moderate constructivism that serves as a basic principle in LEActiveMath. IMS LD on the other hand describes ordered activities in learning and the roles of the involved parties. It is a very broad approach and not well suited for representing a table of contents. In contrast, the purpose of IMS CP is the exchange of content and sequences through the content. Its organization element can be mapped directly to an OMDoc omgroup element and vice versa, and we are working on an IMS CP export of a LEActiveMath table of contents. Still, for the LEActiveMath internal exchange of table of contents, the omgroup element is the best choice because of the system-wide support for OMDoc.

Figure 2.25 shows a table of contents generated using the scenario “LearnNew” with the target concepts “Definition of the average slope” and “Definition of the derivative function”.

2.4.3.4 Distinguishing Between Dynamic and Stable Information

In the discussion of the general challenges of using HTN-planning in a distributed information system, we described that some information may change frequently while other remains stable...
over a longer period of time. This distinction becomes relevant in case performance matters. Accessing resources over the Web often is costly because of high latency. This cost becomes even more of a problem if a mediator links the application and resources because its mapping and query expansions are costly. A solution we investigated is to cache as much information as possible. Depending on whether the information remains stable or is dynamic, the cache spans over a single or over several planning runs.

In our context, caching consists of storing the results of external function calls and re-using them in case the same query arises a second time. This method assumes that the information retrieved in the function call will not change within the time of a single planning run, an assumption that is not unrealistic. The cache is cleared based on application dependent heuristics, for instance after each planning run or after a longer period of time.

In LeActiveMath for instance, information about the learner is stored for a single planning run. Because the learner waits for the course generation to be completed (he sees a progress bar), his state will not change. However, if he later requests a new course to be generated, then presumably his mastery is different. Therefore, in a subsequent planning run, the cache is cleared. In contrast, information about available learning objects is considered to be stable as long as the LeActiveMath system is running. This caching takes place in the mediator and is only cleared when the system is restarted. It is straightforward to add the functionality that allows a resource manager to inform the mediator when its information changes, so that the mediator clears its cache.

2.4.3.5 Integration of the Course Generator

In the following we will describe the integration of the Course Generator into the overall system. LeActiveMath components that need to use the functionalities of the Course Generator do not know (and do not need to know) that a HTN-planner assembles the content. Therefore, the integration of the Course Generator abstracts from the underlying problem-solving framework. Figure 2.26 gives an overview on the classes used to access course generation.

The class Task (bottom right in the figure) represents a pedagogical task as described in Section 2.2.1. It consists of a pedagogical objective and a list of learning object identifiers. The class TutorialInteraction brings together a task and a course (CourseStructure) that supports the learner in achieving the task. A course structure is a structured sequence of identifiers and additional elements, such as templates for natural language generation.

The class CourseGenerator manages tutorial interactions. It creates tutorial interactions and, using the method getCourseStructure, triggers the process of course generation. The actual course generation is implemented by the class JShop2Planner, which inherits the method solve from the interface Solver. Using this interface, we abstract from the framework used for course generation.

The class TutorialControl manages access to the Course Generator: depending on the context, it allows or denies access to course generation. Section 2.8 describes it in detail.

2.4.3.6 Summary

The HTN-framework is well suited for representing complex pedagogical knowledge, especially if it is described hierarchically. Its method and operators are not too hard to understand for non Artificial Intelligence experts; we made the experience that they can be explained easily to pedagogists. The represented knowledge is complex enough that courses can be generated automatically, without human intervention. Similarly, the expressiveness of operators and methods can be used to integrate learning services. By representing the pedagogical knowledge independent
Figure 2.26: An overview on the classes relevant for course generation
from the learning objects, the strategies become re-useable. The scenario “LearnNew” that we described in detail and the other scenarios implemented for LEActiveMath are applicable to other areas of mathematics, too. For instance, although the strategies were developed for calculus, we were able to apply them to content dealing with fractions.

The HTN-framework is efficient, too. To calculate a course that teaches about five concepts on twenty pages takes about ten seconds on a 2.8GH Intel Pentium 4 CPU with 2GB RAM. If afterwards a course covering the same concepts is generated, it takes about 800 milliseconds. This impressive speed-up is achieved by the caching of the mediator. The mediator additionally allows an easy integration of third-party content, as long as there exists a mapping between the mediated ontology and the content.

In Section 2.4.1.2, we described challenges that arise from applying HTN-planning in a course generator, or more general, in a distributed information system. Similar problems are reported by [37], who use AI-planning for Web-service composition. In contrast to their solution, which relies on extending HTN-planning to asynchronous query services, our solutions are built “inside” of the planning framework: they use the provided functionalities in order to address the challenges, but do not modify the algorithm itself. Therefore, properties regarding completion and correctness as described in [17] are not affected.

### 2.5 Supporting the User’s Initiative

In addition to the generation of course that consists of adaptively selected learning materials and learning supporting tools, the Tutorial Component offers two basic features that support the learner’s active engagement in accessing the content: user-triggered course generation and course extension.

In LEActiveMath, course generation is started by the learner. A wizard guides the learner first through the selection of the scenario and then through the selection of the content goals. The assumption is that this way the learner becomes aware of his learning goals and trains to articulate them (see D20, [47]).

Moreover, the explicit representation of tasks and the abstract layer they introduce offers to the learner new ways to access the pedagogical knowledge of the Course Generator. If the learner...
wishes to see additional learning material about a concept in course, she can trigger the execution of a tutorial task (e.g., train) by selecting them from a drop-down list. Then, the task is processed by the Tutorial Component, the resulting learning objects are presented to the learner and integrated in a course upon her request. Figure 2.27 contains a screenshot of the interface.

This approach has the advantage that content is retrieved in a more sophisticated manner than search, sparing the learner the exact knowledge of the learning object metadata. Hence, she can easily and actively request additional content, and thus, a course less resembles a traditional text book but a dynamic and extensible workbook.

2.6 Exercise Sequencer

This section describes the Exercise Sequencer, a component that presents to the learner a dynamically selected sequence of exercises that leads her towards a higher competency level.

This functionality differs from the exercise selection of the course generator: the course generator generates a sequence of learning objects which is adapted to the learner at generation time, but once it is generated, it then remains static. This behavior was a design decision targeted at avoiding confusion of the learner arising from pages changing over and over again as reported in [4].

In contrast, the Exercise Sequencer is completely dynamic. It selects an exercise, presents it to the learner in a GUI separate from the current course, and depending on the learner problem-solving success provides feedback and terminates or selects a new exercise, thus starting the cycle again. The selection algorithm is based on competency levels.

D16, The Integrated Assessment Tool [48], has a similar functionality, in that both tools present a sequence of exercises. However, the goal and thus the underlying theoretical foundations are different. The Assessment Tool is an adaptive test delivery tool which serves to obtain accurate student knowledge estimations in the shortest possible time. Choosing a test item happens based on the Item Response Theory. In contrast, the learning goal of the Exercise Sequencer is to lead the learner to the next higher-competency level.

2.6.1 Using Competency Levels for Exercise Sequencing

In LEACTIVEMATH, competency levels are used to provide information about characteristics of learning objects as well as about the learner. The competency level of a learning object describes the minimum cognitive abilities a learner must possess in order to solve the exercise or understand the learning object with a sufficiently high probability. In PISA (and in similar performance studies [15, 72]), this probability is set at 62%. To put it differently, a learner to whom a competency level $cl$ is assigned to, is able to solve an exercise with competency level $cl$ with a probability of about 62%.

The overall goal of the Exercise Sequencer is to lead the learner from his current competency level to the next one. In the following, we will describe the selection algorithm (illustrated in the flowchart of Figure 2.28).

The Exercise Sequencer is started by the learner: she clicks on a link in a generated course and, like any other learning supporting service, a new window opens in which the sequencer provides information about the goal of actions to come.

Then, the sequencer requests the course generator to select an exercise adequate to the current competency level of the learner using the pedagogical task trainWithSingleExercise!. The task reuses the pedagogical knowledge described in Section 2.4.3, thus following the same principles.
Figure 2.28: An exercise selection algorithm based on competency levels
and avoiding different ways of exercise selection. The sequencer presents the exercise selected by the course generator to learner by delegating it to the exercise system.

When the learner completes working with the exercise, the exercise sequencer receives the event `exerciseFinished`. Depending the success rate it provides positive (`successRate > 0.7`) or neutral feedback (`successRate ≤ 0.7`).

The Exercise Sequencer continues to present exercises adequate to the current competency level until a threshold of 75% of correctly solved exercises has been reached. If the threshold is reached, then exercises of the next higher competency level are trained: the sequencer provides feedback that congratulates the learner and explains the goals of the next sequence of exercises. An additional condition checks whether the learner has reached the highest competency level. In this case, because the level is well mastered and can not be increased, the exercise sequence terminates.

Otherwise, the sequencer lets the course generator choose harder exercises (realized by the task `trainWithHarderExercise`). Again, the sequencer provides feedback depending on the success rate of the learner. This sequence continues until a second threshold of 50% correctly solved exercises is reached. Then, the learner has reached a sufficient mastery at the next higher competency level he has started with. Finally, the sequencer utters summarizing feedback (“You improved your mastery considerably”) and terminates the sequence.

The thresholds of 75% and 50% are motivated by PISA and similar large scale studies [15, 72], which indicate that the probability of successfully solving exercises can serve to distinguish between learners at different competency levels. A recurring claim is that a 62% success rate is sufficient to classify students at specific competency level groups.

To goal of the first phase of the Exercise Sequencer is to ensure that the student is indeed at the assumed competency-level. Because of the short time-span that lies between learning (reading about the content) and evaluation (solving exercises), we increased the threshold from 62% to 75%.

The second phase aims at guiding the learner towards the next higher competency level. We decided for a one-level increase with a threshold of 50% because it is not realistic to increase it even higher in a single lesson without overtaxing the student. Moreover, in contrast to a “pure” assessment situation, the learning process will continue seamlessly as the student moves ahead working through the course (or generates a new course).

### 2.7 Suggestion Agent

This section describes the reactive Suggestion Agent in LeActiveMath. The Suggestion Agent monitors the user’s interactions and offers feedback in case problems are diagnosed. Suggestions consists of navigational hints or learning materials. For the latter case, the Suggestion Agent uses course generation but controlled by the Tutorial Component.

This section addresses the cognitive foundations and the architecture of the Suggestion Agent with its components. This architecture separates components for diagnosis from components for suggestions and different types of information: the observable facts about the student, implied non-observable diagnoses about the student’s actions, suggestions implied from the observable and non-observable diagnoses, and the actual rendering of the suggestions.

#### 2.7.1 Local and Global Feedback

In most intelligent tutoring systems (ITSs) feedback is an immediate reaction to the actual problem solving and supposed to help students to accomplish a solution of an exercise. Usually, this
local feedback reflects the experience with typical errors in the domain of the exercise. In LeActiveMath, this type of feedback is given in the local tutorial dialogue in exercises [49].

More generally, two kinds of feedback and guidance can be provided by an ITS, a local response to student activities which it coaches; the correction of a problem solving attempt of the learner; and a global feedback coaching (several aspects of) the entire learning process. This differentiation somewhat resembles the distinction of task-level and high-level described in the process model of [2]. Local and global feedback differ with respect to which of the student’s (sequences of) activities is supported, scaffolded, or re-assured; what the feedback is about and whether domain-level or meta-level reasoning and learning is targeted and the point in time they are given.

In more detail, local and global feedback differ in several dimensions:

**Realm and Content**  As opposed to the local feedback which essentially informs the student about the (in)correctness of problem solving steps (KR) and may also provide more elaborate hints and even the correct solution, the global feedback may scaffold the student’s overall learning process by commenting on her navigation in the learning material or suggesting additional learning material including examples and exercises.

**Objectives** The purpose of local feedback is a support for a student in problem solving, by correcting a particular solution by giving hints, or by guiding her along a pre-determined problem solution, whereas global feedback is not concerned with one particular exercise and its solution.

**Frame of Student’s Mind** When the student’s mind is (hopefully) focused on solving a particular problem she receives local feedback, whereas before she starts a new learning activity she may receive global feedback.

**Immediate vs. Delayed** Local feedback is provided immediately after each problem solving step or after a few steps. Instead, the global feedback and suggestions can be provided independently of an exercise and may be delayed, i.e., delivered, when the user has finished reading a text, studying an example, or working on an exercise. There is some evidence that the delay may improve the effect of the global (meta-cognitive) feedback. Mathan and Koedinger [55] review diverging results ([59], [11]) on the efficiency of immediate and delayed feedback. One of the possible implications of the divergence suggests that the effectiveness of timing of feedback depends on the nature of the task and of the actual learning process and on the capability of the learner.

**Empirical Findings** Extra-instructional elaboration (EIE) may include a variety of information, e.g., strategic help ([14]), meta-level feedback such as prompts for self-explanation ([9], [8], [80], [75]) or prompts for summarizing and elaborating.

### 2.7.2 Global Feedback in LeActivemath

In order to react to a student’s action, the system has to recognize it in the first place. Then, the action has to be evaluated relative to the learner model, because the same action can have a different meaning for different students. For instance, absolute reading time alone does not allow a proper diagnosis, since this would also depend on the optimal reading time of the student. Currently, the global feedback offers rather general suggestions for navigation through the hypertext material, for the repetition of insufficiently understood concepts and suggests additional exercises. When more elaborate diagnoses and ideas for scaffolding are arising from new insights and empirical facts, then the suggestion component can easily be extended by introducing new rules. This extensibility is due to the modular architecture and declarative representation of rules.
2.7.2.1 Architecture

In LEActiveMath, the diagnosis is separated into two steps. The first step, basic diagnosis, considers observable aspects of the student’s action, e.g., reading time for an item, performance in a problem solving attempt. A second step, diagnostic reasoning, reasons on the diagnostic facts and infers non-observable conclusions. For instance, it reasons that an exercise was solved incorrectly because a prerequisite is unknown. Once the actions have been analyzed, the system figures out how to react to them in the suggestion reasoning step. Suggestions depend on the learner model. For example, the reactions for a very self-confident student may differ from those offered to a student that is not confident at all. Finally, the system communicates the reactions to the student. The actual presentation in the suggestion presentation step depends on the physical device the learner is using and on preferences, e.g., presentation displaying different learning companions. Figure 2.29 illustrates the top-level process flow.

This analysis of the suggestion process led us to an architecture reflecting the five stages as separate components running on a LEActiveMath server.

- The Action Capturing component informs the server about the student’s actions.
- The Basic Diagnoser performs an analysis of the observable values of the student actions. Update information for the mastery assessment in the learner model is generated in this component, too.
- The Diagnostic Reasoner is responsible for a more complex analysis of the student’s behavior and reasons about non-observable diagnoses such as the deficiencies and misconceptions the learner has, whether she got lost in hyperspace, etc. It combines several basic diagnoses.
- The Suggestion Reasoner, reasons about the diagnoses in order to generate suggestions adapted to the student’s current situation.
The Suggestion Rendering component presents the suggestions to the student in a fashion suitable for her.

Each of these components has a well-defined interface in order to provide information to other components. Figure 2.30 shows a more detailed representation of the suggestion engine’s architecture. Currently, the Diagnostic Reasoner and Suggestion Reasoner are implemented using the JESS ([19]) rule engine. Now, each of these components is discussed in more detail below.

**Action Capturing** The generation of global feedback requires monitoring of several types of learning activities. The information about the student’s navigation, her reading, understanding, and problem solving actions serves as a basis for the adaptive suggestions. The action information currently available in any configuration of LeActiveMath is the following time-stamped input for the Basic Diagnoser component:

- start reading a page
- end reading a page
- start working on an exercise
- finished an exercise
- correctly or incorrectly solved exercise

All that information is also stored in the Learner History [51] and stays accessible for further use by the Basic Diagnoser and Diagnostic Reasoner components.

**Basic Diagnoser** This component receives its input from the Action Capturing and performs an evaluation of the student’s actions. As we are interested in different aspects of the student’s actions, there are several basic evaluators which pass the observable behavior input. They can connect to the LeActiveMath knowledge base, e.g., to access metadata, to generate some simple diagnoses from the reported student actions. This processing of the student’s observable action information yields the basic diagnoses.

The collection of evaluators is extensible in order to allow for a more expressive input from the Action Capturing component. Therefore, the architecture is devised in such a way that it is easy to add evaluators and that they are even exchangeable at runtime. The interface to an evaluator contains one entry point for each input the Action Capturing component can deliver and enables each of the basic evaluators to return a specific diagnosis. Not all of the evaluators need to consider all possible inputs. The basic diagnoses are passed on to the Diagnostic Reasoner for further treatment.

**Diagnostic Reasoner** Starting from the basic diagnoses, the Diagnostic Reasoner infers more complex diagnoses from several basic diagnoses and additional information from LeActiveMath’s knowledge base (e.g. metadata) and information from the extended learner model (e.g. whether a concept has already been seen). The complex diagnoses are written into the same fact base as the basic diagnoses and may be used by Diagnostic Reasoner rules to infer additional complex diagnoses. All entries keep a record of the information that was used during the inference as a justification for their existence. This record can be used later on to explain why a suggestion was made and thereby increase the transparency of the system. The Diagnostic Reasoner uses a rule engine (expert system shell) processing declarative knowledge. The rules model human diagnostic reasoning. We describe a set of rules in Section 2.7.2.2.
Suggestion Reasoner  Based on the available diagnostic information, the Suggestion Reasoner reasons for the tutorial reactions. This component is another expert system, and it can use information from the learner model and the knowledge base to conclude suggestion objects. The expert system’s rules model reactions to the diagnoses the Diagnostic Reasoner delivers. The declarative representation of the rules supports the encoding of different reactions to the diagnoses according to differing tutorial strategies. The objects contain several parts: a summary, a feedback message or possibly just an abstract speech act characterization from which natural language feedback can be generated, and one or more actions. Such an object is abstract in the sense that it does not fully determine how the suggestion will be presented to the student.

Suggestion Renderer  In LEActiveMath, the assembly of content and its presentation rendering are separate processes. For the Suggestion Agent, once a suggestion has been generated, it still needs to be presented to the student. This makes the actual presentation more easily adaptable to the student’s needs and context. The actions included into the suggestion objects can be non-verbal as well, e.g. pointing to a page in the table of contents, highlighting information, presenting some learning objects or tasks. Furthermore, the rendering decouples the presentation of a summary from the presentation of full actions. The main benefits of this separation are:

- the feedback stays minimally invasive, i.e. the student receives a hint about what she could do but is not forced to pay attention to it.
- As some of the actions are quite expensive in terms of computation (presentation of content can require course generation), the system does not needlessly waste computational resources, if the suggestion is not requested by the student.

More details and examples for all the components are described in what follows. We first describe the diagnostic process, consisting of both, the Basic Diagnoser and the Diagnostic Reasoner, then the Suggestion Reasoner is discussed, and finally the Suggestion Renderer is presented. In particular, some implemented rules illustrate how the reasoning works for a particular pedagogical scenario.

Notation  As the Diagnostic Reasoner and Suggestion Reasoner make use of the JESS rule engine, we shall use a simplified version of the JESS syntax for presenting the rules. A ? marks variables and (x.y) denotes a call of the method y of an object x. A special object enabling the rules to access frequently needed methods is the ?diagnoseAssistant. It is also used to dynamically access information about the learner, e.g., whether the student has already seen a specific item.

2.7.2.2 Diagnostic Process

The current Diagnostic Reasoner works with the rules that are briefly described and formally presented in the following. Each evaluator in the Basic Diagnoser watches one of a student’s action such as navigation, reading (time), problem solving (assessed performance).

These evaluators can return the following types of facts: presentation of items and pages, and result of a problem solving attempt.

If the user skipped pages she did not see yet, assert a corresponding fact.

IF (pagePresented ?n) THEN ?last = (?diagnoseAssistant.getLastPageSeen) if (?last < (?n-1)) then
(assert (non-contiguous-navigation (page ?last)))
If an exercise about an unmastered concept was presented to the user, but she ignored it, assert a corresponding fact.

\[
\text{IF} \quad \text{ItemPresented (type "exercise")}
\quad \text{(for ?concept)}
\quad \text{(id ?id)}
\quad \text{(onPage ?n)}
\quad \text{(competency ?concept LOW)}
\text{THEN} \quad \text{if (not(?diagnoseAssistant.hasExercise BeenStarted(?id))) then}
\quad \text{(assert (skipped-exercise (page ?n)))}
\]

If an exercise was solved poorly and the user’s competency about a prerequisite concept is low, mark it as missing prerequisite.

\[
\text{IF} \quad \text{ExerciseResult (id ?id) (rating POOR)}
\quad \text{(prerequisite ?concept ?id)}
\quad \text{(competency ?concept LOW)}
\text{THEN} \quad \text{(assert (missing-prerequisite ?n))}
\]

If an exercise was not solved correctly, but the competency values do not seem to be too low, check whether the last \(n\) exercises were also not solved correctly. If that is the case, assert a corresponding fact. This can be used later on to give reassuring feedback.

\[
\text{IF} \quad \text{ExerciseResult (id ?id) (for ?concept) (rating POOR)}
\text{THEN} \quad \text{if (?diagnoseAssistant.lastExercisesFailed(?id)) then}
\quad \text{(assert (lastExercisesFailed ?concept))}
\]

On the other hand, if an exercise was solved correctly, check whether the last \(n\) exercises were also solved correctly. If so, assert a corresponding fact that can be used to challenge the user on a harder exercise.

Figure 2.30: Overview of the Suggestion Agent’s architecture
IF (ExerciseResult (id ?id) (for ?concept) (rating GOOD))
THEN if (?diagnoseAssistant.lastExercisesSolved(n)) then
    (assert (lastExercisesSolved ?concept))

2.7.2.3 Suggestion Reasoner

The Suggestion Reasoner generates abstract suggestions. Abstract means that the actual presentation of the suggestion (the wording, the presentation format, etc.) is not yet determined and is generated only afterwards. These suggestions concern:

- remediation by presenting single learning objects or composed learning material.
- navigation help
- motivating suggestions

The abstract suggestions consist of several slots:

- an abbreviation (e.g., smile/no smile) shown in top line of the rules below
- a suggestion text or speech act shown in the next line of the rule below
- one or more actions following the 'Action:' in the rules below.

Figure 2.31 displays a rendered suggestion.

Rules for Navigation Suggestions  It is known that navigation in hypertexts needs special attention ([61]) because being lost in hyperspace puts an additional cognitive load on the learner. Since LEACTIVEMATH delivers a hypertext learning document, some suggestions deal with navigation. The following describes two rules for navigation suggestions.

If the user skipped some pages in the table of contents she did not see yet, starting from page number \( n \), then propose to navigate back to page \( n \).

IF (non-contiguous-navigation (page ?n))
THEN "You skipped pages that you did not see yet"
  Action: Pointer to page ?n

If the user skipped an exercise on page \( n \) about concepts she is still unfamiliar with, propose to go back to that page and try an exercise.

IF (skipped-exercises (page ?n))
THEN "On the last page, you skipped an exercise that might be useful."
  "What about giving some of them a try?"
  Action: Pointer to page ?n

The action is rendered as a pointers to the table of contents which help the student to return to where she got lost originally.
**Rules for Content Suggestions**  These rules are needed when the target goal-level of competency is not yet reached by the learner. Then studying relevant content and practicing appropriate exercises might help to improve the mastery. The global feedback does not correct single problem solving steps but prompts the learner for activities based on the provision of learning objects. Activities may include also remediating, self-explaining, contrasting, varying, information gathering that are known to improve learning. Here are some rules which address the presentation of learning objects. The rendering can determine the actual prompts.

If a missing prerequisite was diagnosed, present it to the user.

IF  (missingPrerequisite ?concept ?exercise)
THEN  "You should have a look at some prerequisites required to solve that exercise"
      Action: executePedagogicalTask(insertPrerequisites! (?concept))

If the goal concept of an exercise was diagnosed as being insufficiently mastered, then present an exercise training the concept.

IF  (insufficientlyMastered ?concept)
THEN  "You should have a closer look at the concept the exercise is targeting"
      Action: executePedagogicalTask(teachSingleConcept! (?concept))

The rules for Content Selection re-use the pedagogical knowledge represented in the Course Generator: the action only specifies the pedagogical task that can be used to address the diagnosed problem; the actual Learning Objects to be presented are determined by the Course Generator. This ensures a coherent pedagogical approach and avoids that different components use different content selection policies.
**Comforting and Challenging Rules** These rules react to the user’s exercise achievements. The first rule provides reassuring feedback for a user that did not manage to get a good score in the last exercise attempts and provides an easier exercise for the same concept the last failed exercise was for.

\[
\text{IF} \quad (\text{lastExercisesFailed} \ ?\text{concept}) \\
\text{THEN} \quad \text{"Come on, don’t lose hope! We found one more exercise for you to try!"} \\
\text{Action: executeTutorialTask(trainWithSingleExercise!(?concept))}
\]

The goal of this rule is to comfort the user and raise her self-confidence by giving reassuring textual feedback and proposing an easier exercise. In contrast, if the user solves several exercises in a row, it can be useful to compliment on her performance and challenge her by presenting a harder exercise about the same concept the last exercise she solved was about.

\[
\text{IF} \quad (\text{lastExercisesMastered} \ ?\text{concept}) \\
\text{THEN} \quad \text{"Very good! We have prepared a harder exercise for you – can you solve it, too?"} \\
\text{Action: executeTutorialTask(trainWithHarderExercise!(?concept))}
\]

### 2.7.2.4 Conflict Management

Conflicting suggestions may occur in the suggestion fact base. For instance, when a new page is selected by the user, this can trigger a navigation suggestion as well as a conflicting concept presentation suggestion. If not all of the suggestions can be presented at the same time, then a rating has to indicate the priorities, and only the rules with the highest rating will generate their suggestions. For the decision it makes a difference whether two suggestions are both of the same class, say navigation suggestion, or not. Therefore, the rules are classified as navigation, content, and exercise-reaction rules, currently. A conflict between two rules from the same class is resolved by a default resolution (e.g., more specific rules have higher rating). A conflict between rules from different classes is resolved based on a particular conflict resolution strategy. The simplest strategy decides according to a fixed priority rating of the classes. If the highest rated rule cannot generate a suggestion, e.g., because there is no counter example, then it is not executed and the rule with the next highest rating gets a chance.

### 2.7.2.5 The GUI and the Suggestion Rendering

We advocate minimally invasive presentation of the feedback which does not disturb the learning-process. Therefore, the suggestion engine’s window does not pop on top of the screen when a new suggestion is available. The presentation-actions are rendered as links that make the detailed presentations available to the student. If she chooses to have no feedback, she can leave the suggestion window closed. We decided to put links only because we want to give the student the initiative, the choice and the control of whether she wants to see and follow the suggested content. Moreover, there is evidence that some students get frustrated, if the system gives too many hints.

If the user enables the Suggestion Agent, the system displays a light bulb in the menu bar, which is off. When a suggestion becomes available, the bulb is switched on and glows in one of three different colors according to the suggestion type (see Figure 2.32). This allows the user to quickly recognize what the suggestion is roughly about, and to decide whether it is worth to take a look at it or not. If the user wants to see the suggestion, she can click on the light bulb to open the suggestion window. It contains the following elements. The textual suggestion part is rendered as
verbal feedback. Presentation-actions, i.e. presentation of learning objects (examples, exercises, definitions etc.), navigation pointers or highlightings are rendered as buttons which the learner can click to see the actual action.

A suggestion data structure also contains an expiration date that can be used by the system to remove it from the interface and to turn off the light bulb, because the student may focus on something completely different by then. A short history of suggestions is available to guarantee that the student does not lose the opportunity to follow a once presented link just because a suggestion expired.

We are planning the evaluation of the Suggestion Agent with its current set of rules. If it should reveal that the quality of generated suggestions is not good enough, we could probably enhance their quality by making use of the values from the situation model. The wording of suggestions could be adapted to the current user by using her autonomy and approval values. These values could also have an impact on whether a suggestion is actually made. For instance, a user with a high autonomy value might require less suggestions about her navigation.

To summarize, the Suggestion Agent separates information in its modular architecture: the observable facts about the student, implied non-observable diagnoses about the student’s actions, suggestions implied from the observable and non-observable diagnoses, and the actual rendering of the suggestions. Some of the cues and ways that have proved useful by empirical investigations are (partly) realized by the Suggestion Agent. We refer to knowledge familiar to the student, global feedback is sparsely given, brief, and relevant. For the selection of remediating learning material, the Suggestion Agent re-uses the pedagogical knowledge represented in the Course Generator.

### 2.8 Tutorial Control: Managing other Components Access to the Course Generator

The Tutorial Control regulates other components access to the Course Generator. Basically, it needs to cope with two different situations: concurrent access to the Course Generator and context-dependent restrictions.

#### 2.8.1 Concurrent Access of LeActiveMath Components to the Course Generator

In case several clients want to access the Course Generator simultaneously, the Tutorial Control decides which request to consider first, based on a set of rules:

1. Requests from the learner have the highest priority. In line with the pedagogical principles of moderate constructivism, the active engagement of the learner needs to be supported, which declining her requests would contradict. Hence, her requests take precedence.
2. Requests that arise as a direct reaction to an action of the learner are assigned a medium priority. Typical examples are the OLM presenting an exercise to resolve a challenge uttered by the learner; and the assembly tool collecting content in reaction to a request of the learner.

3. Requests from the Suggestion Agent have the lowest priority. These requests arise as an indirect reaction to the learner’s progress and hence “direct” request are described above are preferred.

2.8.2 Context-Dependent Access of LeActiveMath Components to the Course Generator

In addition to the above rules, access to the Course Generator can be denied completely depending on the context of the learning situation. In LeActiveMath, the context of course is given by the scenario used for its creation. D20 [47] specified six scenarios: “LearnNew”, “Rehearse”, “GetOverview”, “TrainCompetency”, “Workbook”, and “ExamSimulation”.

The first five scenarios do not restrict access. In contrast, the “ExamSimulation” aims at imitating an exam situation as realistic as possible. Hence, access to additional content as provided by the course generator is inhibited. Please note that this does not contradict moderate constructivism. The learner explicitly decided to enter this scenario, thus the imposed restrictions are in line with his learning goals. Also, the learner can always leave the course by going back to the main menu of LeActiveMath.

The Tutorial Control potentially could use the autonomy and approval values provided by the situational model in order to perform a more fine-grained ordering of access to the course generator. See the discussion on open issues for more details (Section 5.2).

2.9 Summary

In this section, we described the Tutorial Component, one of the central components of LeActiveMath responsible for adaptivity and learner support. The Tutorial Component embodies complex pedagogical knowledge that allows automatic collection of learning objects and diagnose of learner need, yet is sufficiently efficient for real-world usage. Because each sub-component covers specific pedagogical activities, a coherent and uniform pedagogical approach is possible by reusing pedagogical knowledge wherever possible.

In the following chapter, we will describe how the Tutorial Component is integrated in the LeActiveMath system.
Chapter 3
Integration into LeActiveMath

This chapter describes the integration of the Tutorial Component into the LeActiveMath system. General aspects of the integration such as interfaces and events are covered in the first section. The second section provides details specific to the integration of other LeActiveMath components.

3.1 General Integration

Two ways of integration of the Tutorial Component into LeActiveMath need to be distinguished: a) components requesting the services of the Tutorial Component; b) the Tutorial Component (or its sub-components) using information or services provided by other LeActiveMath components. Sections 3.1.1 to 3.1.4 describe the first case, Section 3.1.5 and 3.1.6 the latter.

3.1.1 Interfaces of the Tutorial Control

LeActiveMath components can access the Course Generator via the Tutorial Control in two ways, either by a classical request-response communication or by delegation.

Request-response access is realized by the following interface:

- public OJDocument handleTask(String userID, String pedaObj, List contentIds)
- public OJDocument handleTask(String userID, String pedaObj, List contentIds, List excludedItems)
- public OJDocument handleTask(String userID, Task task)

These methods implement a synchronous access: a client calls the Tutorial Control with a username, a task (or the two constituents of a task), and optionally a list of identifiers to be excluded by the Course Generator. The result is an OJDocument (the data-structure representing an OMDoc grouping element), which contains the generated course or is empty in case no course fulfilling the task was found. The result can also be null if the request can not be processed because the rules governing access to the Course Generator defined in Section 2.8 decline the request.

The delegation scenario implements a asynchronous access (illustrated in Figure 3.1), in which a client can transfer to the Tutorial Component the complete job of selecting and presenting content:
These methods offer the same functionality as those above, with the difference that the result is an integer value \( n \). For \( n < 0 \), access to the Course Generator was denied; for \( n = 0 \), no course was found; and for \( n > 0 \), a course was found and \( n \) serves as the identifier of the tutorial interaction fulfilling the task.

After the client received the result, it can confirm that the Tutorial Component can proceed in presenting the selected content attached to the tutorial interaction \( n \). This additional interaction is required in case the client wants to perform actions between the positive answer of the Tutorial Component and the actual presentation of the content.

Finally, after the Tutorial Component has presented the course attached to the tutorial interaction \( n \), it sends the corresponding event that informs the client (see the following section for a description of the events).

An additional method serves to retrieve the course attached to a tutorial interaction:

\[ \text{public OJDocument getCourse(int taskID)} \]

### 3.1.2 Interfaces of the Course Generator

The Course Generator is accessible by two methods:

\[ \text{public OJDocument achieveTask(String userID, Task task)} \]
String tutorialInteractionId | The identifier of the tutorial interaction
String userId | The identifier of the user whose actions triggered the event
String pedagogicalObjective | The identifier of the pedagogical objective that the tutorial interaction addresses
List contentIdentifiers | The list of identifiers of the content goals that the tutorial interaction addresses

Table 3.1: Attributes of the TutorialInteractionCourseGeneratedEvent and TutorialInteraction-SeenEvent

- public OJDocument achieveTask(String userID, Task task, List excludedItems)

These methods are used by the Tutorial Control and the exercise sequencer. Furthermore, remote procedure calls using an XML-RPC version of achieveTask allow third-party clients to employ the Course Generator. This was demonstrated in a cooperation with the project TEAL (Task-embedded e-learning), which targets context-specific proactive information delivery for workflow learning. There, we were able to access the Course Generator from the TEAL system and use it to assemble content stored in TEAL’s repository.

3.1.3 Events

Communication by events happens by subscription and publication. A client interested in a specific event type can subscribe to the corresponding tag. Events emitted by the Tutorial Component bear the tag TutorialComponentEventTag.

The Tutorial Component publishes two events, TutorialInteractionCourseGeneratedEvent and TutorialInteraction-SeenEvent (see Table 3.1 for their attributes). The former event is published as soon as a course that achieves the tutorial interaction was generated. The later event is published if in the delegation scenario the content of a course was presented to the learner.

3.1.4 Public Pedagogical Tasks

LEACTIVEMath offers a description of the pedagogical tasks that can be used for communication between components. It specifies the tasks the Course Generator can process (called public tasks), together with additional details necessary for the correct processing.

A public task description consists of the following:

- the identifier of the pedagogical objective;
- the number of concepts the pedagogical objective can be applied to. A task can either be applied to a single concept (cardinality 1) or multiple concepts (cardinality n);
- the type of learning object (as defined in the ontology of instructional objects) that the task can be applied on;
- the type of course to expect as a result. Allowed values are either course in case a complete course is generated or section in case a single section is generated;
- an optional element condition that can be efficiently evaluated in order to determine whether a task can be achieved. In some situations, a service only needs to know whether a
<tasks>
  <task>
    <pedObj id="learnNew"/>
    <contentIDs cardinality="n"/>
    <applicableOn type="fundamental"/>
    <result type="course"/>
    <condition/>
    <description>
      <text xml:lang="en">Generate a course that teaches the concepts to a new learner.</text>
      <text xml:lang="de">Erstelle einen Kurs für Neulerner.</text>
    </description>
  </task>
  <task>
    <pedObj id="explain!"/>
    <contentIDs cardinality="1"/>
    <applicableOn type="fundamental"/>
    <result type="section"/>
    <condition>(class Remark) (relation isFor ?c)</condition>
    <description>
      <text xml:lang="en">Explain the concept.</text>
      <text xml:lang="de">Erkläre das Element</text>
    </description>
  </task>
  <task>
    <pedObj id="illustrateWithSingleExample!"/>
    <contentIDs cardinality="1"/>
    <applicableOn type="fundamental"/>
    <result type="section"/>
    <condition>(class Example)(relation isFor ?mbaseId)(property hasLearningContext ?LearningContext0)</condition>
    <description>
      <text xml:lang="en">Illustrate the concept.</text>
      <text xml:lang="de">Veranschauliche den Inhalt.</text>
    </description>
  </task>
  <task>
    <pedObj id="trainWithSingleExercise!"/>
    <contentIDs cardinality="1"/>
    <applicableOn type="fundamental"/>
    <result type="section"/>
    <condition>(class Exercise)(relation isFor ?mbaseId)(property hasLearningContext ?LearningContext0)</condition>
    <description>
      <text xml:lang="en">Train the concept.</text>
      <text xml:lang="de">Übe den Inhalt.</text>
    </description>
  </task>
  ...
</tasks>

Figure 3.2: A selection of public pedagogical tasks

task will be achieved, but not by which learning objects. There, the value of the condition can be passed to the mediator, and in case of a return value different from null, the task can be achieved. An example is the item menu (Section 2.5) that allows the learner to request additional content. Menu entries are displayed only if the corresponding tasks can be achieved. Technically, the condition element specifies the most relaxed requirements of the corresponding pedagogical methods. Hence, the Course Generator can guarantee that at least a learning object that fulfills these conditions will be returned, but the exact elements will be determined only on-demand.

- a concise natural language description of the purpose that is used for display in menus.

Figure 3.2 contains a selection of public pedagogical tasks. The top element describes the pedagogical task LearnNew. It is applicable to several learning objects of the type fundamental. The bottom element specifies the task trainWithSingleExercise! It is applicable on a single learning object of the type fundamental and will return a result in case the given condition holds.
3.1.5 Dynamically Generated Elements in a Table of Contents

LeActiveMath uses the OMDoc elements omgroup and ref for representing tables of contents:

- the omgroup element serves to provide structure in a table of contents. Two cases are allowed: Either an omgroup represents a section of a course and contains only omgroup elements. Or it consists of ref elements. In this case, it represents a page in a course.

- the ref element references a learning object. The value of its xref attribute contains the identifier of the object to include in the page.

This approach works well in case a course only consists of previously authored learning objects. Yet, some elements of a course depend on the context of the course and thus can not be realized by using pre-authored objects only. This includes calls to learning supporting services, bridging texts, and dynamic tasks.

Therefore, we introduced the new OMDoc element dynamic-item that serves as a generic container for these dynamic cases. These elements are included in a table of contents in the same way as ref elements, but instead of pointing to a learning object they contain the information necessary to generate items on-demand. These items are presented on a page in the same way as referenced learning objects; hence assuring that the learner will not notice any difference.

The general structure of the element dynamic-item is defined by the DTD shown in Figure 3.3. It has a type specifying whether the item is a dynamic task, a call to a learning supporting service or a symbolic representation for text generation. The attributes servicename and queryname allow to further differentiate the specific item to be generated by providing the exact service and method of the service to be called. The optional children of a dynamic-item element specify information about the context: relevant learning objects (using the ref element), mathematical terms in OpenMath format (OMOBJ), and additional parameters given as property-value pairs (queryparam).

The following section provides information about how dynamic elements are used for the integration of learning supporting services in general. Section 3.2.3, 3.2.4 and 3.2.5 describe in detail the integration of the local tutorial dialogue, the concept mapping tool, and the Open Learner Model, resp. Their usage for text generation is topic of Section 3.2.7.

3.1.6 A Generic Integration of Learning Supporting Services

Learning supporting services are software tools that provide the user with information that enhances his learning process. In the LeActiveMath project, several learning supporting service were newly developed and existing ones extended. This section focuses on how these services are integrated in generated courses.
The pedagogical methods encode the knowledge at what time during the learning process the usage of services is most beneficial (see Figure 2.23 for a method that supports the reflection phase). In order to stimulate the learner using services, links are inserted in the generated courses together with a short textual invitation. From the learner’s side, these links do not look different than other interactive elements such as exercises.

Technically, the integration happens the following way:

1. During planning, the Course Generator applies the operator
   \[
   (!\text{insertLearningService} \text{ serviceType queryName ids params})
   \]
   Consequently, later in a course a link is generated that applies the learning service of the type serviceType with the method queryName on the learning objects with the identifiers ids and additional parameters param. For instance, \((!\text{learningService}! \text{ cmap display def slope (isRequiredBy 1)})\) causes a concept mapping tool to display the concept def slope and the learning objects that are required by it.

2. After a plan was found, the above operator triggers the creation of the OMDoc element dynamic-item that represents the above service call. In the element, the identifier of the specific component that implements the requested service replaces the more general service type. This element is then inserted at the current position in the table of content.

3. When the learner visits a page that contains a dynamic item, the presentation system converts the dynamic item into the requested output format (e.g., HTML) and displays it. The rendered element looks the same as any other interactive element: it uses the same layout and is started by clicking on a link, too. This approach was possible without modification of the presentation pipeline and is a nice example of its generality.

By separating the service type from the concrete system used for service provision, the pedagogical knowledge of the Course Generator becomes independent of the actual configuration of the overall system. Hence, the services can be exchanged without affecting the Course Generator, at least as long as the very basic pedagogical principles remain the same.

### 3.2 Integration of LeActiveMath Components

This section provides details on the integration of the Tutorial Component with other LeActiveMath components: the eXtended Learner Model (Section 3.2.1, the Situational Model (Section 3.2.2), the Local Tutorial Dialogue (Section 3.2.3), the Interactive Concept Mapping Tool (Section 3.2.4), the Open Learner Model (Section 3.2.5), and the Assembly Tool (Section 3.2.6). Section 3.2.7 describes the generation of symbolic representations used for text generation.

#### 3.2.1 Extended Learner Model

In order to realize courses adapted to the learner’s competencies and preferences, the Course Generator intensely uses information represented in the extended learner model (xLM).

The access happens on-demand during the course generation by using an external function call instead of mirroring the user information in the world state of the Course Generator.

The external function accesses the information represented in the extended learner model (xLM) via a bridge that hides the concrete xLM implementation. This bridge takes the queries from the Course Generator and builds the corresponding believe descriptors (see Figure 3.4 for a code snippet that generates a belief descriptor accessing metacognitive information).
Figure 3.4: The bridge to the Extended Learner Model

3.2.2 Situational Model

Situation modeling places the individual learners’ requirements in a specific context of an interaction and provides information about the learners’ need for autonomy and approval. Autonomy is defined as the dimension of learners’ face which refers to their need to be allowed the freedom of initiative to discover knowledge by themselves. On the other hand, approval is defined as a learner’s need to have his motivation and emotional balance maintained explicitly by the tutor.

In LEActiveMath, the situational model varies the strength of the recommendations, depending on the scenario of a course. For example, autonomy and approval values will have little impact on the learner’s selection of the particular competency on which to focus in the “Train-Competency” scenario, even if high autonomy is recommended. In this case, control will be with the LEActiveMath system. In the “ExamSimulation” scenario any choices that relate to the selection of competency and difficulty levels of exercises will be limited, i.e., the strength of the situational model’s recommendations will be increased. In “Workbook”, “LearnNew” and “Rehearse” scenarios the strength of recommendations will be generally decreased.

3.2.3 Local Tutorial Dialogue

The local tutorial dialogue application is a dialogue system that can be used to support interactive exercises in a flexible and user-adaptive fashion. The current implementation supports tutoring differentiation exercises by allowing students to practice exercises step-by-step, and offering tutorial feedback adapted to the previous dialogue history and the user model.

The integration of the local tutorial dialogue follows the scheme described in Section 3.1.6: it can be activated by clicking on a link inserted into the learner’s course by the Course Generator during book generation. The URL points to a Java applet that provides the DM’s user interface, and is used by the Tutorial Component to pass parameters related to interaction. The parameters related to the Tutorial Component are “mode” and “taskStatement”. If `mode=exploration`, the dialogue will run in exploration mode, where the local tutorial dialogue GUI (GLEAM) and the server-side DM software are activated without a fixed task statement, and the student can freely set any term as task statement. If `mode=exercise`, the dialogue manager is activated with a fixed task statement.

3.2.4 Interactive Concept Mapping Tool

An additional tool that supports the learning process is the interactive Concept Mapping Tool iCMap (Deliverable D28). iCMap helps the learner to reflect on his mathematical knowledge.
by providing a framework for the visualization and construction of structures in a mathematical domain. It supports the learning process by verifying the concept map constructed by the learner and by suggesting reasonable changes to the created map. Solving a concept map exercise corresponds to completing a partial concept map. Such an exercise is either written by an author or can be constructed by an adaptive and dynamical specification by the Course Generator.

In order to create dynamic exercises, iCMap takes the following parameters from the Course Generator as input: a set of OMDoc identifiers pointing to the initial mathematical concepts to be displayed (the *central concepts*), a set of pairs *(relationType, deepness)*, and the mode-string, which can be *solve* or *display*. Roughly speaking, the concept map contains the initial concepts $C$ and all other concepts that are connected to elements of $C$ by the given set of relations up to the given distance. More precisely:

**Central Concepts** For each learning object defined as a central concept all related learning objects are added to the concept map exercise. The relations taken into account are specified in the parameter *relationType*.

**Relation Type and Deepness** The parameter *relationType* represents the relation types which are used to compute the additional learning items to be presented to the learner. The accepted relations are defined in the ontology of instructional objects. A deepness parameter is attached to each specified relation representing the maximum distance iCMap will follow to compute neighboring learning items. Each node $N$ that is added in the concept map meets one of the following conditions:

1. $N$ is a node representing a central concept, or
2. a relation type $r$ with deepness $r_n$ is defined, such that $N$ is connected by the relation $r$ over at most $r_n$ nodes.

**Mode** The mode specified with the parameter *queryname* determines how the concept map exercise will be presented to the learner. If the mode is *display* all the computed nodes and all the possible edges of the given types are added to the workspace and the learner is told to verify the map and, if applicable, to complete it. Launching an exercise with mode *solve* starts iCMap with an empty concept map. All the nodes determined as central concepts and all those computed by iCMap are added to the learners palette. In mode *solve* the learner has therefore to create the concept map by herself.

Figure 3.5 illustrates a plan operator, the dynamic element, the rendered exercise and the resulting workbench of an iCMap exercise.

### 3.2.5 Open Learner Model

#### 3.2.5.1 The Open Learner Model as a Learning Supporting Service

The Open Learner Model (OLM) is integrated in a course as a learning supporting service. Thus, links to the OLM are inserted in a course at places determined by the pedagogical methods using the operator

```
(!insertLearningService OLM display ids descriptor)
```

The parameters *ids* and *descriptor* are used to construct a belief descriptor that is initially displayed in the OLM workbench: *descriptor* is a list of strings that consists of the identifiers of the metacognitive, motivational, affective, and competency dimension of a belief descriptor. *ids* is a list of learning object identifiers and is used to describe the subject domain dimension.
Figure 3.5: An example of the integration of the concept mapping tool
Deliverable D24
Tutorial Component

Figure 3.6: An example of integration of the Open Learner Model

During plan application the operator triggers the insertion of a dynamic item at the given location in the table of contents. Finally, when the learner visits the page, the presentation system inserts a link that starts the OLM with the given belief descriptor. Because the OLM uses topics to represent the subject domain, the learning object identifiers encapsulated by the link are replaced by topic identifiers.

Figure 3.6 illustrates the OLM-integration: a plan operator application, the generated dynamic item, and the link in the course. In the example, the generated belief descriptor contains the top node of the competency dimension that includes all the 8 mathematical competencies.

3.2.5.2 Suggestions to the Tutorial Component

When the negotiation between the Open Learner Model and the learner on a topic fails, the Open Learner Model has the possibility to suggest to perform more exercises before resuming the discussion (the assuming that additional evidence may lead either the Open Learner Model or the learner to change their position on the disagreement).

Such a possibility is offered by the Tutorial Component and provides with an interesting bridge between the tutorial and self-reflective aspect of LEACTIVEMATH.

In discussion with Edinburgh and Glasgow it turned out that it was unclear whether the expected value of task delegation in case of the Open Learner Model and the local tutorial dialogue would offset the implementation effort. As it is not critical to the overall system, we assigned it a low priority and will realize it, if possible.

3.2.6 Assembly Tool

LEACTIVEMATH’s assembly tool (Deliverable D37) allows a learner to manually create his own book and to upload it into LeActiveMath. This process supports the learner’s meta-cognitive reasoning, helps to memorize knowledge, and to work with content actively.
The assembly tool’s main actions are to create book structures by adding chapters, to drag-and-drop learning items from a browser LeActiveMath is running in, and to retrieve and add notes associated with single learning items (see Figure 3.7). Created books can be transmitted to LeActiveMath and are then presented in the ‘My Books’ space. Each user-generated book, whether created by the assembly tool or by the course generator, can be opened for further editing in the assembly tool. The assembly tool integrates the Tutorial Component by connecting to its service and asking for further content for a particular learning item.

The integration is realized as follows: upon a right-mouse click on a definition item a context menu opens. It offers actions that map to tasks provided by the tutorial component. After selecting an action, the assembly tool sends a request to the Tutorial Component service including the affected learning item identifier, the task name, and the user identifier. The Tutorial Component generates a new table of contents and sends it back. Finally, the assembly tool maps the items from the table of contents to its internal structure and inserts them just below the learning item for which the request is invoked.

### 3.2.7 Symbolic Representations for Text Generation

Transitions, introductions, and summarizations make transitions between learning objects smoother and help to understand the structure and learning goals of courses and sections.

In LeActiveMath, the learning goals are represented in the pedagogical tasks and methods of the Course Generator. Thus, the Course Generator is the adequate component to create symbolic representations used for text generation.

During the planning, the Course Generator can use the operator

\[(!text \text{textType} (id_1, \ldots, id_n))\]

to trigger the creation of a symbolic representation for a text of type \text{textType} about all learning objects with identifier \(id \in (id_1, \ldots, id_n)\). After plan transformation this results in the following dynamic item added in the table of contents:

```xml
<dynamic-item type=“text” servicename=“NLGGenerator” queryname=“textType”>
  <ref xref=“id_1” />
  ...
  <ref xref=“id_n” /> 
</dynamic-item>
```
The example in Figure 3.8 shows all three stages used in text generation. Please note that in the current preliminary implementation the learning object identifiers are ignored.

Text generation is also required for creating/inserting section titles. There, titles are generated during plan transformation and inserted as metadata of the `omgroup` element the plan is transformed to.

Following the scheme described above, the arguments of the operator (`!startSection textType (id1, ..., idn)`) determine the generated text, which is inserted in the element `metadata/title` of the `omgroup` element that represents the current section. For every supported language, a title is generated. The example in Figure 3.9 illustrates the title generation for a section that serves as an introduction to the “Definition of Average Slope”.

---

Why is the mathematical content presented in this book important? The following section tries to answer that question.

Figure 3.8: From a planning operator to a template to a generated text

Figure 3.9: Generating section titles
Chapter 4

Evaluations

Evaluations are an essential part of the development of the LeActiveMath environment. Accordingly, several constituents of the Tutorial Component were evaluated as soon as they were available. This section describes the results of the evaluations. We assessed the postulated general applicability of the ontology of instructional objects (Section 4.1), the performance of the Course Generator and mediator (Section 4.2.2 and 4.2.1). In addition to the more technical studies, we assessed in a first formative evaluation the quality of the scenario LearnFlow as perceived by students (Section 4.3) and performed a lab-based cooperative evaluation (Section 4.4). Section 4.5 describes the further studies planned by Munich to assess the Tutorial Component.

4.1 Evaluations of the Ontology Of Instructional Objects

One design goal of the ontology of instructional objects was compatibility with existing knowledge representations. Therefore, we analyzed to what extent the ontology could be mapped to several selected representations:

- DocBook [89] serves a standard for writing structured documents using SGML or XML and was selected for the evaluation because of its wide-spread use. Its elements describe the complete structure of a document down to basic entities, e.g., the parameters of functions. Here, the most relevant elements are those that describe content at paragraph level (called “block” elements).

- WINDS ([78]), a Web-based Intelligent Design and Tutoring System, uses the adaptive learning environment ALE to provide several adaptive hypermedia features, e.g., adaptive link annotation. Its knowledge representation is based on Cisco’s learning objects [10] and was selected because its design was explicitly influenced by pedagogical considerations.

- For the same reason, we included the “Multidimensional Learning Objects and Modular Lectures Markup Language”, <ML>³ ([52]). Several German universities used it to encode about 150 content modules.

- DaMiT (http://damit.dfki.de) is an adaptive tutoring system teaching data mining [31]. It adapts to the individual learning style of a user by providing different views (e.g. formal vs. informal) on the same learning content.

- MATHSTHESAURUS (http://thesaurus.maths.org) is an online multilingual Mathematics Thesaurus in nine languages [85] and was selected because it covers a wide range of mathematics.
By and large, we were able to design mappings between the ontology of instructional objects and third-party knowledge representations. Most problems were raised by the fact that often third-party elements had no instructional semantics (e.g., para in DocBook, quotation and description in <ML>\(^3\)). For additional information about the mappings, we refer the reader to [86].

DaMiT and MathsThesaurus were integrated for usage with the Course Generator: In a cooperation with the project TEAL, we devised a mapping and integrated the DaMiT database using the mediator. As a result, the TEAL system was able to use the Course Generator. In the same way, the content of the MathsThesaurus was integrated.

Additional evaluations focused on pedagogical considerations. Several school teachers (for mathematics and physics), instructional designers and members of e-learning lab of Klett were questioned about domain independence, pedagogical flexibility, completeness, and applicability. The feedback was largely positive, and suggestions (e.g., an additional class “LawOfNature”) were taken into account for a revision of the ontology.

Applications of the ontology in domains other than course generation were investigated in the European Network of Excellence Kaleidoscope and published in [58]. Additionally, it was recently used for a revised version of the ALOCoM ontology, a recent effort in the European Network of Excellence ProLearn [34], in the e-learning platform e-aula [74], and in the CampusContent project of the Distant University Hagen [36].

## 4.2 Technical Evaluations of the Tutorial Component

### 4.2.1 Performance of the Course Generator

One of the major shortcomings of the former Course Generator (besides of inflexibilities that made it impossible to represent complex pedagogical expertise) was its limited performance in case several learner used it simultaneously. Therefore, as soon as the new Course Generator was available we compared the efficiency of both approaches.

In order to be able to compare the old, expert system based (EXSCG) and the new course generator (HTNCG), we used scenario GuidedTour which was available in the old system and re-implemented in the current system. The tests and all necessary components (ActiveMath server and a learning object repository) ran on a 2.8GH Intel Pentium 4 CPU with 2GB RAM. Both course generators repeatedly generated an average length course about the mathematical concept derivation.

EXSCG showed a slightly slower performance for single user course generation (average time of 23 vs. 20 seconds). However, the EXSCG course generation did not terminate in case of the simultaneous generation of more than 20 users. HTNCG slowed down, but still terminated (average of 2 minutes).

We suspect the differences to be caused by the on-demand retrieval of the metadata. In EXSCG, the metadata of the complete content was loaded in the fact-based of the expert system and then the rules were evaluated. In HTNCG, content relevant for the current plan is retrieved if necessary, i.e., when queried.

Please note that this evaluation took place at an early stage of implementation, and neither the mediator nor the Course Generator had a caching mechanism implemented. It was encouraging that the new approach was faster even though an additional indirect step via the mediator was inserted.

In the study, we were able to test some effects of caches by using a cached version of the LeActiveMath learning object repository. There, in case the very same query is posed more than once,
the result elements were not retrieved from index but directly from memory. Course generation for a single user decreased to the average time of three seconds, which is about factor 7 faster than compared to the non-cached version. The speedup holds in case of 30 user planning, which took about 20 seconds. This speedup is even more surprising in regard of the fact that due the early stage of implementation the cache was local to a single user, which means instead of one “global” cache, the system used up to 30 “local” caches. We expect the speedup to be caused mainly by the large amount of similar queries, which occur even in a single course generation. For instance, if several examples for the same concept are inserted in a course, a large number of the same queries are processed.

A final part of the study compared an XML-RPC based connection and a direct connection between the mediator and the LeActiveMath learning object repository. Because of the overhead of processing XML-RPC, we expected the direct connection to outperform XML-RPC. Surprisingly, the direct connection proved not to be much faster. A possible cause may be the limited number of queries (about 270).

### 4.2.2 Performance of the Mediator

In the description of mediator, we suggested that a cache might yield considerable performance gains, especially as the Course Generator often queries for learning objects with the same or with similar metadata.

To determine whether this is indeed the case, we performed an evaluation with varying cache sizes. This study took place at a later time than the above study. The results are illustrated in Figure 4.1 and support the efficiency of the cache. “Duration, first run” stands for the time a course generation takes on a freshly started up LeActiveMath system. “Duration, second run” represents the time required for a subsequent course generation, on the same pedagogical task (scenario “LearnNew” with the concept “Definition of the derivative function”). The small decrease even without a cache is caused by the cache of the repository. Larger cache sizes accelerate the planning dramatically.

Still, performance remains an issue. During the development of the Tutorial Component, the implemented pedagogical strategies became more and more complex, hence requiring more reasoning, and as a result, decreasing the system’s performance. See “Open Issues” (Section 5.2) for additional details.

<table>
<thead>
<tr>
<th>Cache size</th>
<th>0</th>
<th>20</th>
<th>500</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of queries to the mediator</td>
<td>505</td>
<td>505</td>
<td>505</td>
<td>505</td>
</tr>
<tr>
<td>Duration, first run</td>
<td>10.1 s</td>
<td>8.2 s</td>
<td>4.9 s</td>
<td>4.9 s</td>
</tr>
<tr>
<td>Duration, second run</td>
<td>9.8 s</td>
<td>5.6 s</td>
<td>3 s</td>
<td>0.8 s</td>
</tr>
<tr>
<td>Number of queries to MBase, first run</td>
<td>10,838</td>
<td>1218</td>
<td>1041</td>
<td>947</td>
</tr>
<tr>
<td>Number of queries to MBase, second run</td>
<td>10,838</td>
<td>1103</td>
<td>852</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 4.1: Planning a course with learning objects from MBase using different cache sizes.
4.3 First Formative Evaluations in Saarbrücken and Munich

A first formative evaluation aimed at assessing the scenario “LearnNew”. Two groups of students were asked to work through the content, one group using a generated course, the other using a predefined course. We wanted to determine whether the generated books were perceived to be useful by the learners and whether they make the learning process more efficient by the adaptive selection of content.

The evaluation took place both at the universities in Munich and in Saarbrücken. In Munich the subjects were five students for lectureship with mathematics as one of their subjects and in their final year of study. In Saarbrücken six computer science students attending a seminar on “e-learning and Mathematics” participated in the evaluation.

In both places the procedure was as follows: At first an introduction, which lasted approximately 15 minutes, was given to the students. It consisted of an overview of the content (chapters, sections and subsections) and the navigation and assistance features. The different exercise formats were demonstrated briefly.

Afterwards in each location the students were randomly split in two groups: the CourseGeneration-group (CGG) and the PredefinedBook-Group (PBG). The students in the CGG worked with a generated course, the PBG work with a manually assembled book that contained the complete content. That way, we hoped to be able to assess whether an automatic selection of the learning objects proves to yield an advantage over being able to access the complete content within a single book. Each group had additionally access to the search facility [46].

The task for the subjects was to refresh their knowledge starting with the concept of the derivative up to the differentiation rules. The task duration was about 45 minutes. Afterwards each student completed a questionnaire that assessed attitudes (e.g., like and dislike of the course they worked with) and how they judge the effectiveness of the course.

The majority of questions was formulated as statements that requested a closed answer in the form of an agreement with one of four levels between “exactly” and “not at all” of a Likert scale. Some open questions were put into the questionnaire as well, e.g. the students were expected to name advantages or additional problems they could think of for first time learners.

For the quantitative analysis, the single statements of the subjects were encoded numerically, with the allowed values ranging from 1 to 4. Approval for the system as a whole resp. for single component is expressed in low values. Assertions meeting the same aspect were grouped into categories. Those categories are overall positive views on the system, quality of navigation, adequacy of contents and value for new learners. The means were calculated for different groups and compared. Open statements (qualitative analysis) were also a part of the questionnaire. They have been collected and categorized (according to specific features of the system).

As a result regarding the overall positive view on the system, the students gave average valuations of 2.54 as a mean (range: 1 to 4), which indicates a positive view on the system. The lower and therefore better values were given by the students in Saarbrücken (2.31) (Munich: 2.82).

Several question inquired about the perceived usefulness of the course for first time learners. Here the mean was 2.85. As expressed in the feedback the students considered the content or system as a whole to be to complicated for such learners. Contrary to our expectations we found no relevant difference between the groups. The quantitative results were in general confirmed by the qualitative ones.

A closer look can explain the unexpected results of the study. First of all, the generated courses were indeed not optimal. Especially the exercise, example, and proof selection did not sufficiently
take into account the required prerequisites and difficulty level. As a result, we adapted the formalization of the pedagogical strategies.

Even more problematic were the existing mathematical skills of the subjects, which were much lower than we expected. Their problems started with concepts we considered being prerequisites, and hence, in the limited time the subjects were not able to reach the content goals of the courses. This certainly influenced the comparison of pre-recorded and generated courses, because some students didn’t even reach those pages that were adapted.

A third factor were technical problems with the system. Because the students had to work with a recent version of the Tutorial Component, we could not resort to a stable server but were forced to use a developer version, in which some components proved to be unstable.

Still, the study did point out some problems with the pedagogical strategies. We will evaluate the effects of the adopted changes in a future study.

4.4 Lab-based Evaluation in Edinburgh

During February and March 2006 a lab-based University-level evaluation of LEActiveMath was performed. The system was evaluated without the advanced forms of the Tutorial Component, Course Generator, or xLM. By evaluating the system without these components a clear baseline could be established and used to identify the potential contributions of the Tutorial Component to the student’s learning and experience of LEActiveMath.

A cooperative evaluation design was used for this study. This design actively engages the student in the evaluation process and enables them to step back and critique the system’s performance and comment on their experience. Students were set tasks to complete and asked to "think aloud" describing how they are carrying out the task and any problems they are having. During their interaction audio and video was recorded and later analyzed to identify metrics of performance. This data was then combined with the student’s answers to pre- and post-use questionnaires.

Eleven students (6 male; mean age 19.45 years) from University of Edinburgh first year Mathematics courses took part in the evaluation. The participants had been studying calculus for 2.6 years on average and rated their own confidence with calculus as "good". All participants were frequent computer users who were very familiar with web interfaces. All had previously used some form of maths software although the tasks they had performed were mostly limited to generating graphs and inputting mathematical formulae. These participants were considered representative of the University-level target users of LEActiveMath.

All content in the primitive version of LEActiveMath evaluated was presented to the student in a non-reactive form. Content was divided into either prerecorded courses which had been constructed by a teacher or personal courses which the student could construct themselves using the old Course Generator. Due to the absence of reactive components and xLM the content presented in these courses was the same for all students and did not react in anyway to the student model. Essentially the student’s experience of the content was as an electronic text book with hyperlinked concepts.

The evaluation showed that students found the structure and navigation of the content "quite" easy to use and "quite" useful (responses were made on a 5-point Likert scale with "quite" being the equivalent of 4). 91% of students said that the book metaphor used to structure the content was useful and intuitive and the web interface worked as expected. However, 63% of the students commented that the content structure became confusing when there were more than two sublevels of content. When a course was subdivided into a top level (chapters) and a bottom level (pages), students navigated the content without any problem. If a course contained further subdivision...
they were unable to use the book metaphor to refer to the content and their efficiency of navigation suffered.

In the primitive version evaluated the only way the student could tailor the content to their preferences was to construct their own course. This involved following a wizard and selecting content topics to appear in the course. On average students rated the Course Generator as "quite" easy to use but of only "moderate" usefulness. When asked for more details 63% of students commented that they found it difficult to relate the list of content items presented in the course creation tool to the resulting form of that content in the generated course. The relationship between content item and page was not one-to-one as expected.

When asked for suggestions about how LEACTIVEMATH could be improved students suggested that they would appreciate step-by-step examples and exercises that had a clear structure. They also expected LEACTIVEMATH to contain revision tests at the end of each chapter and were surprised to find this was absent from the primitive version. When asked to identify LEACTIVEMATH's weaknesses they commented that the system was not as good as a human tutor as it could not direct them towards content and exercises based on their knowledge or give them tailor made guidance. However, no requests for these functions were made when given the opportunity to suggest improvements. This mismatch possibly indicates students' assumptions about what a computer system is capable of. This suggests that the functionality offered by the reactive Tutorial Component in combination with the xLM should be unexpected but beneficial to the student’s experience of LEACTIVEMATH and their learning.

In summary, the evaluation of the primitive version of LEACTIVEMATH was very positive with students finding the system easy to use and liking the functions offered by the system. However, the lack of reactivity and personalization of the content limited the potential benefit of the system to the student’s learning. The inclusion of these features in the final version should resolve these issues. Additionally, the wizard used for course generation was completely redesigned. The success and benefit of the reactive tutorial component will be assessed as part of the final classroom evaluations and reported in deliverable D44.

### 4.5 Further Planned Evaluations

Further evaluations are planned by Saarbrücken and Munich as well as by Edinburgh. Munich and Saarbrücken will again compare pre-recorded against generated courses.

The intention of the studies planned by Munich is to use the controlled environment of a laboratory to generate detailed feedback about users’ impressions of the pedagogical strategies, content, exercises, and overall design of the LEACTIVEMATH system. The studies are designed for testing with strong emphasize on the distinctive features of the scenario and on the special requirements of the user group.

As a consequence of the first lab study we will no longer make the subjects (university students) systematically work on the generated courses themselves but give them the task to judge courses. The value and estimated efficiency of applying a pedagogical strategy will be assessed, with respect to the addressed user group.

In addition, several scenarios will be evaluated during the school evaluation. There, we will also ask teachers to complete questionnaires inquiring about their judgments on generated courses.
Chapter 5

Conclusion

5.1 Related Work

Adaptivity in hypermedia (Web-based) systems has a long history. Early approaches on course generation date back to the eighties [68, 62]. With the Generic Tutoring Environment (GTE), [87, 54] vanMarcke introduced the separation between instructional tasks (representations of pedagogical activities) and instructional methods (representations of different ways of achieving the activities). Most of this instructional knowledge is independent of the learning domain. Vassileva [88] build on this approach and added several layers of rules that handled different aspects of course generation (e.g., selection of the main strategy or of the appropriate media types, etc.).

Subsequent course generation frameworks as developed by [79] or [82] did not reach such a high level of representing pedagogical knowledge (or at least never provided sufficiently detailed descriptions).

Today’s course generation focuses less on pedagogical knowledge, but more on Semantic Web and metadata, e.g., [32] on using ontologies of the subject domain to automatically calculate the best path through the learning material, or [33] on calculating a learner specific ranking of and trail through learning objects retrieved according to his query. These approaches use rather simplified pedagogical knowledge, e.g., to select those learning objects with the lowest typical learning time. However, to generate a course which is adapted to the individual learner’s goals and needs and which is based on state of the art pedagogical strategies requires more elaborate expertise.

A related strain of research is Adaptive Hypermedia (for an overview, see [5]). One of its techniques, Adaptive Presentation, allows to conditionally include text fragments. In systems like AHA [1], conditional rules are included into the hypertext document. Using a technique like dynamic task execution allows to move the rules from the document to a dedicated component like the Course Generator with the described advantages.

Open corpus hypermedia as described in [23] allows to integrate HTML pages from different sources. Using information about the concepts a page covers (provided by the author), the system generates a trail (course) leading the learner to her learning goal. The generation takes into account the learner’s knowledge and the dependency relations between the concepts, but no other explicitly represented pedagogical knowledge.

Regarding the mediator approach described in Section 2.4.2, several other approaches for data integration by ontology mapping as well as for federation (i.e., the reuse and exchange) of learning objects exist [65, 13, 67, 60]. However, these approaches were either not yet implemented at the time of design of LEACTIVEMATH or too early in development.
To summarize, it is safe to say that none of the previous work fulfilled the requirement of representing as complex pedagogical knowledge as realized within the LEActiveMath project while at the same time being usable in real-world applications. Additionally, most of former work focused on single aspects of learning support, and none provided an encompassing framework able to integrate such a variant of service like the Tutorial Component of LEActiveMath.

5.2 Outstanding Issues

So far, this report has described the Tutorial Component, starting with the basic principles influencing its design, which include pedagogical principles as well as the decision for a declarative, task-based representation of the underlying pedagogical knowledge. The different constituents that build up the Tutorial Component where explained in detail, their internal communication, mostly based on pedagogical task, and the interfaces making it functionality accessible as a service to other LEActiveMath components and third-party systems. The practical value, that is, performance and pedagogical effectiveness, was evaluated in a first set of studies.

In this final section, we describe some open issues that have been caused by the complexity of the LEActiveMath project and the interdisciplinary communication that is difficult at phases.

Delays due to refinements of the pedagogical strategies We underestimated the efforts that were necessary to revise the pedagogical strategies defined in deliverable D20 [47]. These revisions proved necessary for two reasons: first, even though the descriptions in D20 were quite precise and formal in comparison with common pedagogical guidelines, some issues obvious for pedagogical experts were not specified and, hence, not implemented. These issues were detected as soon as the Course Generator produced the first courses. This led to still ongoing refinements in the formalization. Second, in the first formative evaluation some shortcomings in the example and exercise selection were detected. These, too, led to adaptations of the strategies.

Ongoing implementation of the pedagogical strategies The necessary refinements described above required changing some of the pedagogical methods at a stage, where not all scenarios were implemented. Thus, at the time of writing two of the six scenarios are not yet available (Workbook and ExamSimulation), as well as some advanced features such as the inclusion of user-interface elements (e.g., buttons) in a course that add additional examples or exercises. However, this latter functionality is available indirectly using the item menu described in Section 2.5. The scenario Workbook will re-use the existing methods for exercise selection, and hence we expect to complete its implementation in due time.

In contrast, the scenario ExamSimulation introduces a time constraint for the generated courses, i.e., a limit that the estimated average time required for solving the exercises should not exceed (e.g., 90 minutes). The straightforward approach we intended to follow was to use the backtracking that the planner provides. However, it turned out that as soon as major backtracking is involved, the run time of the Course Generator increases significantly. In general, this does not dramatically affect the remaining scenarios (see the next paragraph), but might prove problematic for the scenario examSimulation.

Course generation performance The current Course Generator is a major improvement over the former Course Generator, with respect to the complexity of the pedagogical knowledge that can be implemented as well as with respect to the performance as the studies in Section 4.2 have shown. However, complexity affects the performance: the more elaborate the implementation of the strategies became, the more reasoning was required, and as a result, the system’s performance decreased.
Currently, a course generation depending on the scenario takes between a few seconds and two minutes. While this seems reasonable taking into account that generated courses may consist of about twenty section with fifty learning objects, for a student waiting for the process to finish even a minute may seem a long time.

A first analysis showed that most of the time is consumed by the backtracking involved in the example and exercise selection. The remaining methods are implemented in a way that most of the time backtracking is avoided. Therefore, we will investigate how the selection process can be optimized.

**Extending pre-recorded books with dynamic elements** Dynamic elements that allow to include calls to learning supporting services and the on-demand execution of pedagogical tasks are new features and hence unfamiliar to authors and teachers.

However, they offer intriguing new possibilities for authors and teachers. An author/teacher can design a course, where parts of her course are predefined, and others dynamically computed taking the learner model into account. In this way, an author can profit from the best of both worlds: she can compose parts of the course by hand and at the same time profit from the adaptive features of the Tutorial Component. Now, with the stable Tutorial Component, we will start exploring these features.

**Further usages of the course generation service** The Course Generator is accessible to other LeActiveMath components via the Tutorial Control. Currently, this service provision is used by the Suggestion Agent and directly by the learner using LeActiveMath’s Web interface. Potentially, other LeActiveMath components can use the Course Generator as well: the local tutorial dialogue for explaining mathematical rules that the learner applied incorrectly; the Open Learner Model for resolving impasses reached during the negotiation phase, and the assembly tool for supporting the learner while creating a personal book.

The required interfaces are defined and partly implemented. The support of delegation is still missing, which allows a component to transfer the complete task of content selection and presentation to the Tutorial Component. In discussion with Edinburgh and Glasgow it turned out that it was unclear whether the expected value of task delegation in case of the Open Learner Model and the local tutorial dialogue would offset the implementation effort. As it is not critical to the overall system, we assigned it a low priority and will realize it, if possible.

**Usage of Situational Model in the Tutorial Control** In the current implementation, the Tutorial Component uses the information about the learner provided by the Situational Model indirectly by accessing the xLM. Direct use was foreseen for regulating priorities of requests in the Tutorial Control. For instance, request of a student with low autonomy might receive a lower priority. Yet, we decided against such adaptive change of priorities, because it would contradict the guideline of putting the learner in control imposed by the moderate constructivism as well as potentially conflict with the learner’s expectations with regard to the behavior of a Web-based system (which, too puts him in control). Another potential usage concerns the Suggestion Agent. If evaluations should reveal that the quality of generated suggestions is not good enough, we could probably enhance their quality by making use of the values from the situation model.

**Topics and Concepts** The xLM uses topics for its internal representation of the subject domain. Topics are abstractions of OMDoc concepts. In some cases, several definitions at OMDoc level might be represented by a single topic in the xLM. In LeActiveMath, this happens if several definitions define the same mathematical concept but for different learning contexts (such as higher education or university first cycle). Thus, these definitions are variants of each other.
Because the Course Generator performs its reasoning over OMDOC elements, situations may arise in which the learner has reached different competency levels depending on the variant of a definition. But as the xLM is not able to distinguish between variants, problems might arise. In LeActiveMath, the only allowed type of variants of concepts are those that vary on the learning context. Because each learner is assigned to a single learning context, problems caused by several concepts being represented by a single topic can be avoided. However, in our view (DFKI and UdS), the problems sustain our claim that the learner model needs a finer-grained representation than topics.

To summarize, none of the above outstanding issues poses a serious threat to the LeActiveMath project. The Tutorial Component is running smoothly in its current state and the remaining issues rather concern fine-tuning than principal issues.
Bibliography


Appendix A

Interfaces and Events

• Interfaces of the Tutorial Control
  – public OJDocument handleTask(String userID, String pedaObj, List contentIds)
  – public OJDocument handleTask(String userID, String pedaObj, List contentIds, List excludedItems)
  – public OJDocument handleTask(String userID, Task task)
  – public int delegateTask(String userID, String pedaObj, List contentIds)
  – public int delegateTask(String userID, String pedaObj, List contentIds, List excludedItems)
  – public int delegateTask(String userID, Task task)
  – public boolean confirmTask(int taskID)
  – public OJDocument getCourse(int taskID)

• Interfaces of the Course Generator
  – public OJDocument achieveTask(String userID, Task task)
  – public OJDocument achieveTask(String userID, Task task, List excludedItems)

• Events published by the Tutorial Component
  – TutorialInteractionCourseGeneratedEvent
  – TutorialInteractionSeenEvent
Appendix B

Planning Domain Description

(defun CourseGeneration

;; basic operators

;; section commands do not cost anything
!(:operator (!startSection ?title)
  ()
  ()
  ()
  0
)

!(:operator (!startSection ?title ?contentIds)
  ()
  ()
  ()
  0
)

!(:operator (!endSection)
  ()
  ()
  ()
  0
)

!(:operator (!!addInWorldState ?atom)
  ()
  ()
  (?atom)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Methods and operators that insert an element in the course

;; The basic operator: add the logical expression that the element
;; was inserted in the course. This will cause the creation of a
;; reference in the generated table of content.

!(:operator (!insertElement ?i)
  ;; precondition
  ()
  ;; delete
  ()
  ;; add
  ((inserted ?i))
)

;; needed because only the operator !insertElement can create references.
!(:method (insertElementOnce! ?i)
 MethodInsertElementOnce!
  ;; If element ?i was not yet inserted, insert it and mark
  ;; it as inserted
  (}
(not (inserted ?i))
)
((insertElement ?i)
)
)

; similar to insertElementOnce but do not fail if the element was
; already inserted.
(:method (insertElement ?i)
MethodInsertElement
;; precondition
{
(not (inserted ?i))
)
((insertElement ?i)
)
MethodInsertElementFallback
()
()
)

; try to insert all elements of the given list
(:method (insertAllElements (?head . ?rest))
MethodInsertAllElements
()
((insertElement ?head)
 (insertAllElements ?rest)
)
)

;; this method ensure that inserting all elements does not fail for
;; the empty list (due to the behavior of JSHOP2, it can not be
;; merged in the above method).
(:method (insertAllElements nil)
MethodInsertAllElementsFallback
()
()
)

; try to insert all elements of the given list if learner is ready
(:method (insertAllElementsIfReady (?head . ?rest) ?concept)
MethodInsertAllElementsIfReadyIfReady
()
 (insertIfReady ?head ?rest)
 (insertAllElementsIfReady ?rest)
)

;; this method ensure that inserting all elements does not fail for
;; the empty list
(:method (insertAllElementsIfReady nil)
MethodInsertAllElementsIfReadyFallback
()
()
)

;; Insert the given element if the learner is/will be ready to
;; understand it.
(:method (insertIfReady ?element ?concept)
 (not (inserted ?element))
 (ready ?element ?concept)
)
 ((insertElement ?element)
 )
(:method (insertIfReady ?element ?concept)
 (inserted ?element)
)
)

;; Insert the given element if the learner is/will be ready to
;; understand it, fail if already inserted or not ready.
;:method (insertIfReady! ?element ?concept)
  (not (inserted ?element))
  (ready ?element ?concept)
  (!insertElement ?element)

;; Insert the given element and all variants of it if the learner
;; is/will be ready to understand it, fail if already inserted.
;:method (insertWithVariantsIfReady! ?element ?concept)
  (not (inserted ?element))
  (ready ?element ?concept)
  assign ?variants
  (call GetElements
    (class InstructionalObject)
    (relation isVariantOf ?element))
    (call GetRelated (?element) -1
      (class InstructionalObject)
      (relation isVariantOf placeholder))
    (!insertElement ?element)
    (addInWorldStateAsInserted ?variants)

;; Two helper methods that iterate over a list and insert every
;; item of the list.
;:method (addInWorldStateAsInserted nil)
  MethodAddInWorldStateAsInsertedEmptyList
  ()

;:method (addInWorldStateAsInserted (?first . ?rest))
  MethodAddInWorldStateAsInserted
  (!addInWorldState (inserted ?first))
  (addInWorldStateAsInserted ?rest)

;:- (ready ?element ?concept)
;; An learner is/will be ready to understand an element if its *for*-values
;; 1. are in the current course, OR
;; 2. are well-known, OR
;; 3. equal to ?concept
;; AND all auxiliaries it does require are in the current course

  (call GetElements
    (class Fundamental)
    (relation inverseIsFor ?element)))
  (call GetElements
    (class Auxiliary)
    (relation inverseIsFor ?element)))

;:- (readyFor nil ?concept)

;:- (readyFor (?first . ?rest) ?concept)
  (or (same ?first ?concept)
    (inserted ?first)
(and (learnerProperty hasCompetencyLevel ?first ?cl)
(call >= ?cl 3))
)

(readyFor ?rest ?concept)
}
)

;; If there are no elements to test, then the learner is ready for
;; the element.
(:- (readyRequires nil)
()
)

;; Otherwise check the first element whether it and the rest of the
;; elements fulfill any of the above criteria
(:- (readyRequires ((?first . ?rest)))
{
(inserted ?first)
(readyRequires ?rest)
}
)

;; Insert a call of a learning service.
() () ()
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; LazyTasks
(:operator (!lazyTask ?educationalObjective ?contentIDs)
; ()
; ()
; ()
; ()
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Testing for achieved sub-goals
;; Mark the given task as achieved.
(:operator (!!setAchieved ?task)
; ()
; ()
; (achieved ?task)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; collecting prerequisites

;; collect prerequisites without taking the educational level
;; into account. Used by guidedTour.
(:- (collectUnknownPrereqIgnoreLearningContext ?c ?result)
;; used by GuidedTour
(,:first (}
(learnerProperty hasEducationalLevel ?el)
(learnerProperty hasField ?field)
;; collect prerequisites
(assign ?prereqs (call GetRelated (?c) -1

  (((class Fundamental)
    (relation isRequiredBy placeholder)
    (property hasLearningContext ?el)
    (property hasField ?field)
  )

  ((class Fundamental)
    (relation isRequiredBy placeholder)
    (property hasLearningContext ?el)
    (property hasField ?field)
  )

  ((class Fundamental)
    (relation isRequiredBy placeholder)
    (property hasLearningContext ?el)
    (property hasField ?field)
  )))

;; sort
(assign ?tempResult (call Sort ?prereqs

  (((class Fundamental)
    (relation isRequiredBy placeholder)
    (property hasLearningContext ?el)
    (property hasField ?field)
  )

  ((class Fundamental)
    (relation isRequiredBy placeholder)
    (property hasLearningContext ?el)
    (property hasField ?field)
  )))

;; remove known or inserted concepts. The result is inverted, hence invert it again.
(removeKnownConcepts ?tempResult2 ?tempResult nil)
(assign ?tempResult3 (call Reverse ?tempResult2))
;(call PrintDebug collectUnknownPrereq tempResult2 ?tempResult2)
(same ?result ?tempResult3)
)
)

(:- (collectUnknownPrereq ?c ?result)
 ;; used by LearnNew
 (:first
  (learnerProperty hasEducationalLevel ?el)
  (learnerProperty hasField ?field)
  ;; get all elements
  (assign ?elements (call GetRelated (?c) -1

    (((class Fundamental)
      (relation isRequiredBy placeholder)
      (property hasLearningContext ?el)
      (property hasField ?field)
    )

    ((class Fundamental)
      (relation isRequiredBy placeholder)
      (property hasLearningContext ?el)
      (property hasField ?field)
    )))

  )

  (not (same ?elements nil))
  ;; sort
  (assign ?sorted (call Sort ?elements

    (((class Fundamental)
      (relation isRequiredBy placeholder)
      (property hasLearningContext ?el)
      (property hasField ?field)
    )

    ((class Fundamental)
      (relation isRequiredBy placeholder)
      (property hasLearningContext ?el)
    ))

  ))

  ;; remove known or inserted concepts. The result is inverted, hence invert it again.
  (removeKnownConcepts ?reversedUnknown ?sorted nil)
  (assign ?unknown (call Reverse ?reversedUnknown))
  (same ?result ?unknown)
)
)
;; ; initialisation
;; ; retrieve the goal and insert a task for it
; (:method (initialise)
  ; ; Retrieve the goal from the logical atom:
  (goalTask ?task)
  (insertGoal ?task)
  )
;; ; Insert the goal as a task.
; (:method (insertGoal (?pedObjective ?contentIDs))
  guidedTour
  (same ?pedObjective "guidedTour")
  (guidedTour ?contentIDs)
  )
illustrateWithSingleExample!
((same ?pedObjective "illustrateWithSingleExample!")
 (startSection ?pedObjective ?contentIDs)
 (illustrateWithSingleExample! ?contentIDs)
 (endSection)
 )
;; ; (class Concept) (relation isRequiredBy ?mbaseId) (property hasLearningContext ?LearningContext0) learnPrerequisitesConceptsGT
((same ?pedObjective "learnPrerequisitesConceptsGT")
 (startSection ?pedObjective ?contentIDs)
 (learnPrerequisitesConceptsGT ?contentIDs)
 (endSection)
 )
;; ; (class Concept) (relation isRequiredBy ?mbaseId) (property hasLearningContext ?LearningContext0) learnPrerequisitesConceptsShort
((same ?pedObjective "learnPrerequisitesConceptsShort")
 (learnPrerequisitesConceptsShort ?contentIDs)
 )
trainWithSingleExercise
((same ?pedObjective "trainWithSingleExercise!")
 (startSection ?pedObjective ?contentIDs)
 (trainWithSingleExercise! ?contentIDs)
 (endSection)
 )
learnNew
((same ?pedObjective "learnNew")
 (learnNew ?contentIDs)
 )
; (class Introduction) (relation isFor ?baseId) (property hasLearningContext ?LearningContext0)
  motivate!
  (same ?pedObjective "motivate!")
  (startSection ?pedObjective ?contentIDs)
  (motivate! ?contentIDs)
  (endSection)
)

; (class Introduction) (relation isFor ?baseId) (property hasLearningContext ?LearningContext0)
  introduceShort!
  (same ?pedObjective "introduceShort!")
  (startSection ?pedObjective ?contentIDs)
  (introduceShort! ?contentIDs)
  (endSection)
)

; (class Remark) (relation isFor ?c)
  explain!
  (same ?pedObjective "explain!")
  (startSection ?pedObjective ?contentIDs)
  (explain! ?contentIDs)
  (endSection)
)

; (class Conclusion) (relation isFor ?c)
  conclude!
  (same ?pedObjective "conclude!")
  (startSection ?pedObjective ?contentIDs)
  (conclude! ?contentIDs)
  (endSection)
)

trainWithIncreasedDiff
  (same ?pedObjective "trainWithIncreasedDiff")
  (!addInWorldState (expand))
  (!startSection lazyTask (?pedObjective))
  (trainWithIncreasedDiff ?contentIDs)
  (endSection)
)

(rehearse ?contentIDs)
)

; overview
initOverview
  (same ?pedObjective "overview")
  (overview ?contentIDs)
)

; train Competency
initTrainCompetencyThink
  (same ?pedObjective "trainCompetencyThink")
  (trainCompetency think ?contentIDs)
)
initTrainCompetencyArgue

(same ?pedObjective "trainCompetencyArgue")
)
(trainCompetency argue ?contentIDs)
)initTrainCompetencyModel
(same ?pedObjective "trainCompetencyModel")
(trainCompetency model ?contentIDs)
)initTrainCompetencySolve
(same ?pedObjective "trainCompetencySolve")
(trainCompetency solve ?contentIDs)
)initTrainCompetencyRepresent
(same ?pedObjective "trainCompetencyRepresent")
(trainCompetency represent ?contentIDs)
)initTrainCompetencyLanguage
(same ?pedObjective "trainCompetencyLanguage")
(trainCompetency language ?contentIDs)
)initTrainCompetencyCommunicate
(same ?pedObjective "trainCompetencyCommunicate")
(trainCompetency communicate ?contentIDs)
)initTrainCompetencyTools
(same ?pedObjective "trainCompetencyTools")
(trainCompetency tools ?contentIDs)

;; a dummy task that always succeeds as long as a
;; contentId is given
dummyTask
(same ?pedObjective "dummyTask")
(first ?id ?contentIDs)
(!startSection dummyTask (?pedObjective))
(!insertElement ?id)
(!endSection)

;; for testing purposes
testTT
(same ?pedObjective "testTT")
(!startSection Test ?contentIDs)
(!startSection think ?contentIDs)
(trainCompetency think ?contentIDs)
(!endSection)
(!startSection argue ?contentIDs)
(trainCompetency argue ?contentIDs)
(!endSection)
(!startSection model ?contentIDs)
(trainCompetency model ?contentIDs)
(!endSection)
(!startSection solve ?contentIDs)
test
{
(same ?pedObjective "test")
}
(!startSection test)
(test ?contentIDs)
(!endSection)
)

testEmptyGrouping
{
(same ?pedObjective "testEmptyGrouping")
}
(!startSection testEmptyGrouping)
(!startSection stillEmpty)
(!endSection)
(!startSection emptyToo)
(!endSection)
(!endSection)
)

;; if no goal is inserted, it presumably is a bug.
initFail
;; print out a message
((call PrintDebug WARNING! NO GOAL INSERTED!))
;; and fail by trying to achieve an unachievable task
((fail))
)

;; for testing purposes
(:method (test ?f)
MethodTest
{
(first ?c ?f)
}
(train! ?c)
; (reflect (?f))
; (illustrateNew! ?f)
)

;;: ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; pedagogical scenarios
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; guided tour
(:method (guidedTour ?concepts)
MethodGuidedTour
() )
(!startSection GuidedTour)
(:method (learnConceptsGuidedTour (?concepts))
  (reflect ?concepts)
  (!endSection)
  )

(:method (learnConceptsGuidedTour (?c . ?rest))
  MethodLearningConceptsByGuidedTour
  ()
  (learnPrerequisitesConceptsGT ?c)
  (learnSingleConceptGT ?c)
  (learnConceptsGuidedTour ?rest)
  )

(:method (learnConceptsGuidedTour nil)
  MethodLearningConceptsByGuidedTourDummy
  ()
  )

;; learning a single concept
(:method (learnSingleConceptGT ?c)
  MethodLearnSingleConcept
  ;; Check whether the task has been achieved.
  (not (achieved (learnSingleConceptGT ?c)))
  
  (!startSection Title (?c))
  (introduceByIntroductionSection ?c)
  (insertConceptSection ?c)
  (explainSection ?c)
  (illustrateWithIncreasedDiffSection ?c)
  (trainWithIncreasedDiffSection ?c)
  (concludeSection ?c)
  (!endSection)
  (!!setAchieved (learnSingleConceptGT ?c))
  )

MethodLearnSingleConceptTaskAchieved
  ;; If the task was achieved, skip it.
  (achieved (learnSingleConceptGT ?c))
  ()
  )

(:method (insertConceptSection ?c)
  ()
  (!startSection Title (?c))
  (insertElement ?c)
  ;; Hack for DAMIT: comment the next line in
  ;; (insertDefinitionFor ?c)
  (!endSection)
  )

(:method (insertDefinitionFor ?c)
  MethodDAMITHack
  (assignIterator ?def
   (call
    GetElements ((class Definition)
      (relation isFor ?c)))))
  (insertElementOnce! ?def)
  )

()()

(:method (insertElementsOfType ?for ?type ?difficulty ?amount)
  MethodInsertElementsOfType
  ()

(c)LEActiveMath Consortium 2006 Page 99 of 143
(:method (insertElementsOfType! ?for ?type ?difficulty ?amount)
  MethodInsertElementsOfTypefallback
  ()
  ()
)

;; check whether there exist any element with the
;; required difficulty. If not, fail.
(:method (insertElementsOfType! ?for ?type ?difficulty ?amount)
  MethodInsertElementsOfType!
  {
    ;; check whether there exist any element with the
    ;; required difficulty and learning context and field
    (assign ?test (call GetElements ((class ?type)
      (relation isFor ?for)
      (property hasDifficulty ?difficulty)
    ));; get the user's educational level and field
      (LearnerProperty hasEduationalLevel ?el)
      (LearnerProperty hasField ?field)
    ;; collect elements of fitting learning context and field
      (getNonInserted ?elements
        (call GetElements ((class ?type)
          (relation isFor ?for)
          (property hasDifficulty ?difficulty)
          (property hasLearningContext ?el)
          (property hasField ?field)
        ))
        (assign ?length (call Length ?elements))
        (call >= ?length ?amount)
        )
      (insertUpTo ?amount ?elements)
    )
    ;; in this case, there are not enough fitting elements
    (assign ?test (call GetElements ((class ?type)
      (relation isFor ?for)
      (property hasDifficulty ?difficulty)
    ));; get the user's educational level and field
      (LearnerProperty hasEduationalLevel ?el)
      (LearnerProperty hasField ?field)
    ;; collect elements of fitting learning context and field
      (getNonInserted ?elements
        (call GetElements ((class ?type)
          (relation isFor ?for)
          (property hasDifficulty ?difficulty)
          (property hasLearningContext ?el)
          (property hasField ?field)
        ))
        (assign ?length (call Length ?elements))
        )
      (insertAllElements ?elements)
  }
)

  MethodInsertElementsOfTypeRelaxed!
  {
    ;; get the user's educational level and field
      (LearnerProperty hasEducationalLevel ?el)
    ;; collect elements of fitting learning context
      (getNonInserted ?elements
        (call GetElements ((class ?type)
          (relation isFor ?for)
          (property hasDifficulty ?difficulty)
          (property hasLearningContext ?el)
        ))
        (assign ?length (call Length ?elements))
  }
)
(call >= ?length ?amount)
)

(insertUpTo ?amount ?elements)
)

MethodInsertElementsOfTypeRelaxed12!
(;; in this case, there are not enough fitting elements
;; get the user's educational level
(learnerProperty hasEducationalLevel ?el)
;; collect elements of fitting learning context
(getNonInserted ?elements
(call GetElements ((class ?type)
(property hasLearningContext ?el)
(relation isFor ?for)
(property hasDifficulty ?difficulty)
))
)

(assign ?length (call length ?elements))
)

(insertAllElements ?elements)

)

MethodInsertElementsOfTypeRelaxed2!
(;; collect elements
(getNonInserted ?elements
(call GetElements ((class ?type)
(relation isFor ?for)
(property hasDifficulty ?difficulty)
))
)

(assign ?length (call length ?elements))

(assign ?newAmount (call - ?amount ?length))
;; if the following fails, then no fitting elements were inserted.
(call != ?newAmount ?oldAmount)
)

(insertUpTo ?amount ?elements)
)

(:method (insertUpTo ?amount ?elements)
MethodInsertUpToBase
((or
(call <= ?amount 0)
(same ?elements nil)))
)

MethodInsertUpTo
(
(first ?first ?elements)
(rest ?rest ?elements)
)

(insertElement ?first)

(insertUpTo (call - ?amount 1) ?rest)
)
)

(:method (learnNew ?concepts)
MethodLearnNew
(
)
(!!startSection LearnNew)
(learnConceptsLearnNew ?concepts)
(!!endSection)
)

):

(:method (learnConceptsLearnNew (?c . ?rest))
MethodLearningConceptsByLearnNew
(!!startSection LearnNew)
(learnConceptsLearnNew ?concepts)
(!!endSection)
)

):
(learnConceptsLearnNew ?c)
  (learnConceptLearnNew ?c)
)

(:method (learnConceptLearnNew nil))
  MethodLearningConceptsByLearnNewFallback
  ()
  ()

(:method (learnConceptsLearnNew ?c))
  MethodLearnConceptsLearnNew
  ()

  ;; for this method to be fulfilled, at least an introduction or the
  ;; prerequisites section needs to be inserted.

  (:method (introduceWithPrereqSection! ?c))
    MethodIntroduceWithPrereqSection!1
    ()
    (introduceWithSection! ?c)
    (learnPrerequisitesConceptsShortSection ?c)
  )

  (:method (introduceWithSection! ?c))
    MethodIntroduceWithSection!1
    ()
    ;; Start a new section for the motivation
    (!startSection Introduction (?c))
    (!text Introduction (?c))
    (motivate !c)
    (problem !c)
    (introductionExampleify !c)
    (!endSection)
  )

  (:method (introduceWithSection! ?c))
    MethodIntroduceWithSection!2
    ()
    ;; Start a new section for the motivation
    (!startSection Introduction (?c))
    (!text Introduction (?c))
    (motivate !c)
    (problem !c)
    (introductionExampleify !c)
    (!endSection)
  )

  (:method (introduceWithPrereqSection ?c))
    MethodIntroduceWithPrereqSection
    ()
    (introduceWithPrereqSection! ?c)
  )

  (:method (introduceWithPrereqSection ?c))
    MethodIntroduceWithPrereqSectionFallback
    ()
    ()

  ;; the following methods construct an introduction. Each method
  ;; differs in the critical task. The methods will fail in case no
  ;; part of the introduction can be satisfied.

  (:method (introduceWithSection !c))
    MethodIntroduceWithSection!1
    ()
    ;; Start a new section for the motivation
    (!startSection Introduction (?c))
    (!text Introduction (?c))
    (motivate !c)
    (problem !c)
    (introductionExampleify !c)
    (!endSection)
  )

  (:method (introduceWithSection !c))
    MethodIntroduceWithSection!2
    ()
    ;; Start a new section for the motivation
    (!startSection Introduction (?c))
    (!text Introduction (?c))
    (motivate !c)
    (problem !c)
    (introductionExampleify !c)
    (!endSection)
  )
(:method (introduceWithSection !?c)
  MethodIntroduceWithSection!3
  ()
  (;; Start a new section for the motivation
  (!startSection Introduction (?c))
  (!text Introduction (?c))
  (motivate ?c)
  (problem !?c)
  (introductionExamplify !?c)
  (!endSection)
  )
)

(:method (introduceWithSection !?c)
  MethodIntroduceWithSection!3
  ()
  (;; Start a new section for the motivation
  (!startSection Introduction (?c))
  (!text Introduction (?c))
  (motivate ?c)
  (problem ?c)
  (introductionExamplify !?c)
  (!endSection)
  )
)

(:method (introduceWithSection ?c)
  MethodIntroduceWithSection
  ()
  ((introduceWithSection! ?c))
)

(:method (introduceWithSection ?c)
  MethodIntroduceWithSectionFallback
  ()
  ()
)

(:method (developConcept ?c)
  MethodDevelopConceptHighCompetencyLevel
  (;; If the learner has a high competency level, insert
  ;; the concept and a single example
  (learnerProperty hasCompetencyLevel ?c ?cl)
  (call >= ?cl 3)
  )
  (!
  (!startSection Title (?c))
  (!text Develop (?c))
  (!insertElement ?c)
  (illustrateWithSingleExample ?c)
  (!endSection)
  )
)

MethodDevelopConceptHighMotivationLowCompetencyLevel
  (;; If the learner has a high motivation but low
  ;; competency level, insert an explanation and
  ;; examples.
  (learnerProperty hasCompetencyLevel ?c ?cl)
  (call < ?cl 3)
  (learnerProperty hasMotivation ?c ?mo)
  (call >= ?mo 3)
  )
  (!
  (!startSection Title (?c))
  (!text Develop (?c))
  (!insertElement ?c)
  (insertIntroductionExercise ?c)
  (explain ?c)
  (illustrate ?c)
  (!endSection)
  )
)

MethodDevelopConceptFallback
  ()
  (;; Otherwise insert the concept, text and examples
  (!startSection Title (?c))
  (!text Develop (?c))
  (!insertElement ?c)
  (explain ?c)
  (illustrate ?c)
  (!endSection)
  )
)
(:method (practiceSection ?c)  
  PracticeSection  
  ()  
  ((practiceSection!? ?c))  
)

(:method (practiceSection ?c)  
  PracticeSectionFallback  
  ()  
  ()  
)

(:method (practiceSection! ?c)  
  MethodPracticeSection!  
  ()  
  ;; Insert exercises.  
  (startSection Exercises (?c))  
  (train! ?c)  
  (endSection)  
)

(:method (connectSection! ?c)  
  MethodConnectCMapSection!  
  (learningServiceAvailable CMap)  
  )  
  (startSection Theorems)  
  (connectByCMap! ?c)  
  (endSection)  
)

MethodConnectShowTheoremsSection!  
()  
  (startSection Theorems)  
  (connectByTheoremWithProof! ?c)  
  (endSection)  
)

(:method (connectSection ?c)  
  MethodConnectSection  
  ()  
  (connectSection! ?c)  
)

(:method (connectSection ?c)  
  MethodConnectFallback  
  ()  
  ()  
)

(:method (connect! ?c)  
  MethodConnectCMap!  
  (learningServiceAvailable CMap)  
  )  
  (connectByCMap! ?c)  
)

MethodConnectShowTheorems!  
()  
  (connectByTheoremWithProof! ?c)  
)

(:method (connect ?c)  
  MethodConnect  
  (connect! ?c)  
)
(:method (connect ?c)
  MethodConnectFallback
  ()
  ()
)

(:method (connectByCMap! ?c)
  MethodConnectByCMap!
  {
    (learningServiceAvailable CMap)
  }
  {
    (text Connect (?c))
    (insertLearningService CMap display (?c) (requires 1 isRequiredBy 1 isA 1 inverseIsA 1))
  }
)

(:method (connectByTheoremWithProof! ?c)
  MethodConnectByTheoremWithProof!
  {
    (learnerProperty hasEducationalLevel ?el)
    ;; retrieve all theorems that require the fundamental ?c
    (assign ?allTheorems
      (call GetElements ((class Law)
        (relation requires ?c)
        (property hasLearningContext ?el)
      )))
    ;; sort the theorems
    (assign ?sortedTheorems
      (call Sort ?allTheorems
        ((class Law)
          (relation isRequiredBy placeholder)
          (property hasLearningContext ?el)))))
    (not (same ?sortedTheorems nil))
  }
  {
    (text Connect (?c))
    (connectTheoremsWithProof ?sortedTheorems)
  }
)

; insert all theorems of the given list
(:method (connectTheoremsWithProof (?head . ?rest))
  MethodConnectsTheoremsWithProof
  ()
  ((connectTheoremsWithProof ?head)
    (connectTheoremsWithProof ?rest))
)

(:method (connectTheoremsWithProof nil)
  MethodConnectsTheoremsWithProofFallback
  ()
  ()
)

(:method (connectTheoremWithProof ?theorem)
  MethodConnectTheoremWithProof
  ()
  {
    ;; insert the theorems (if not already inserted, see,
    ;; e.g., scenario overview)
    (insertElementOnce! ?theorem)
    (remarksForTheorem ?theorem)
    (proof ?theorem)
  }
)

():method (connectTheoremWithProof ?theorem)
  MethodConnectTheoremWithProofFallback
  ()
  ()
)
(:method (connectByTheoremSection ?c) 
  MethodConnectByTheoremSection () 
  (connectByTheoremSection! ?c) )

(:method (connectByTheoremSection ?c) 
  MethodConnectByTheoremSectionFallback () 
  ()
)

(:method (connectByTheoremSection! ?c) 
  MethodConnectByTheoremSection! () 
  (connectByTheoremSection! ?c) )

(:method (connectTheorems (?head . ?rest)) 
  MethodConnectsTheorems () 
  ((connectTheorem ?head) 
    (not (same ?sortedTheorems nil)))

 (:method (connectTheorems nil) 
  MethodConnectsTheoremsFallback () 
  ()
)

 (:method (connectTheorem ?theorem) 
  MethodConnectTheorem ()
  (insertElementOnce! ?theorem) 
  (not (same ?sortedTheorems nil)))

 (:method (connectTheorem ?theorem) 
  MethodConnectTheoremFallback () 
  ()
)

 (:method (connectTheorem ?theorem) 
  MethodConnectTheorem ()
  (insertElementOnce! ?theorem) 
  (not (same ?sortedTheorems nil)))

 (:method (connectTheorem ?theorem) 
  MethodConnectTheoremFallback () 
  ()
)

 (:method (connectTheorem ?theorem) 
  MethodConnectTheorem ()
  (insertElementOnce! ?theorem) 
  (not (same ?sortedTheorems nil)))

 (:method (connectTheorem ?theorem) 
  MethodConnectTheoremFallback () 
  ()
)

 (:method (connectTheorem ?theorem) 
  MethodConnectTheorem ()
  (insertElementOnce! ?theorem) 
  (not (same ?sortedTheorems nil)))

 (:method (connectTheorem ?theorem) 
  MethodConnectTheoremFallback () 
  ()
)
\begin{verbatim}
{ (startSection Reflection)
  (insertLearningService OLM display (?first) (competencyId competency))
  (endSection)
}
MethodReflectManual
;; Otherwise insert a text
()
{
  (startSection Reflection)
  (text Reflect ?concepts)
  (endSection)
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; scenario Rehearse
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:method (rehearse ?concepts)
  MethodRehearse()
  {
    (startSection Rehearse)
    (rehearseConcepts ?concepts)
    (endSection)
  }
)
(:method (rehearseConcepts (?c . ?rest))
  MethodRehearseConcepts()
  {
    (rehearseSingleConcept ?c)
    (rehearseConcepts ?rest)
  }
)
(:method (rehearseConcepts nil)
  MethodRehearseConceptsFallback()
  ()
)
(:method (rehearseSingleConcept ?c)
  MethodRehearseSingleConcept()
  {
    (startSection Title (?c))
    (insertConceptSection ?c)
    (illustrateSection ?c)
    (connectByTheoremSection ?c)
    (practiceSection ?c)
    (endSection)
  }
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; scenario Overview
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:method (overview ?contentIDs)
  MethodOverview()
  {
    (first ?c ?contentIDs)
    (LearnerProperty hasEducationalLevel ?el)
    ;; The overview scenario presents the principal concept
    ;; and how it is connected to other concepts via theorems.
    ;; First get the theorems that require the current concepts.
    (assign ?theorems (call GetElements ((class Law)
                                          (relation requires ?c)
                                          (property hasLearningContext ?el)) ))
    ;; Now, retrieve the definitions upon which the theorems
    ;; depend.
    (assign ?definitionsH
\end{verbatim}
(call GetRelated ?theorems 1
  (class Definition)
  (relation isRequiredBy placeholder)
  (property hasLearningContext ?el)
  )))
  (removeElement ?definitions ?c ?definitionsH)
(call Sort ?definitions
  (class Definition)
  (relation isRequiredBy placeholder)
  (property hasLearningContext ?el)
  )))
)

;; First elaborate the principal concept.
(!startSection Overview ?contentIDs)
(!startSection Principal_Concept (?c))
(!startSection Title (?c))
(insertElement ?c)
(CMapOverview ?theorems)
(!endSection)
(!endSection)
;; Then show its connections to the other definitions.
(!startSection Develop_Connections ?sortedDefinitions)
(developConnections ?sortedDefinitions (?c))
(!endSection)
(CMapOverviewExercise ?theorems)
(!endSection)

(:method (CMapOverview ?theorems)
  MethodCMapOverview
  ;; use the cmap tool if available
  (learningServiceAvailable CMap)
  )
  (insertLearningService CMap display ?theorems (requires 1))
  )
MethodCMapOverviewFallback
;; CMap not available:
()
()
()

(:method (CMapOverviewExercise ?theorems)
  MethodCMapOverviewExercise
  ;; use the cmap tool if available
  (learningServiceAvailable CMap)
  )
  (insertLearningService CMap solve ?theorems (requires 1))
  )
MethodCMapOverviewExerciseFallback
;; CMap not available:
()
()
()

(:method (developConnections (?c . ?rest) ?connected)
  MethodDevelopConnections
  ()
  (!startSection Title (?c))
  (illustrateSection ?c)
  (practiceSection ?c)
  (developConnectionSingleConcept ?c ?connected)
  (!endSection)
  (developConnections ?rest (?c . ?connected))
  )
)

(:method (developConnections nil ?connected)
  MethodDevelopConnectionsFallback
  ()
  ()
  )
(:method (developConnectionSingleConcept ?c ?concepts)
  MethodDevelopConnectionSingleConcept
  {
    (collectConnectedTheorems ?theorems ?c ?concepts)
    (learnerProperty hasEducationalLevel ?el)
    (assign ?sortedTheorems
      (call Sort ?theorems
        ((class Law)
          (relation isRequiredBy placeholder)
          (property hasLearningContext ?el)
          ))))
  }
  {
    (!startSection Connections (?c))
    (!insertElement ?c)
    (connectTheoremsWithProof ?sortedTheorems)
    (!endSection)
  })

(:- (collectConnectedTheorems ?theorems ?c ?concepts)
  (collectConnectedTheoremsH ?theorems ?c ?concepts nil)
)

(:- (collectConnectedTheoremsH ?theorems ?c nil ?temp)
  (same ?temp ?theorems))

(:- (collectConnectedTheoremsH ?theorems ?c (?first . ?rest) ?temp)
  {
    (learnerProperty hasEducationalLevel ?el)
    (assign ?theoremsA
      (call GetElements ((class Law)
        (relation requires ?c)
        (property hasLearningContext ?el)
        ))))
    (assign ?theoremsB
      (call GetElements ((class Law)
        (relation requires ?first)
        (property hasLearningContext ?el)
        )))
    (assign ?intersection
      (call Retain ?theoremsA ?theoremsB))
    (collectConnectedTheoremsH ?theorems
      ?c ?rest
      (call Concat ?temp ?intersection))
  })

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; scenario TrainCompetency
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(:method (trainCompetency ?competency ?contentIDs)
  MethodTrainCompetency
  ()
  {
    (!startSection trainCompetency)
    (trainCompetencyH ?competency ?contentIDs)
    (!endSection)
  }
  )

(:method (trainCompetencyH ?competency (?first . ?rest))
  MethodTrainCompetencyH
  ()
  {
    (!startSection Title (?first))
    (trainCompetencySingleConcept ?competency ?first)
    (!endSection)
    (trainCompetencyH ?competency ?rest)
  } )

(:method (trainCompetencyH ?competency nil)
  MethodTrainCompetencyHFallback
  ()
  )

©LeActiveMath Consortium 2006
:method (trainCompetencySingleConcept ?competency ?c)  
MethodTrainCompetencySingleConcept1  
  (learnerProperty hasCompetencyLevel ?c 1.0)  
  (startSection Rehearse (?c))  
  (insertElement ?c)  
  (endSection)  
  (trainCompetencyExamplesExercises ?competency ?c 1.0)  

MethodTrainCompetencySingleConceptGreater1  
  (learnerProperty hasCompetencyLevel ?c ?cl)  
  (call > ?cl 1.0)  
  (assign ?newCl (call - ?cl 1.0))  
  (startSection Rehearse (?c))  
  (insertElement ?c)  
  (endSection)  
  (trainCompetencyExamplesExercises ?competency ?c ?newCl)  

(:method (trainCompetencyExamplesExercises ?competency ?c ?cl)  
MethodTrainCompetencyExamplesExercises  
  (call < ?cl 4.0)  
  (setCompetencyLevel ?c ?cl)  
  (illustrateCompetencySection ?competency ?c)  
  (practiceCompetencySection ?competency ?c)  
  (!deleteSetCompetencyLevel ?c ?cl)  
  (trainCompetencyExamplesExercises ?competency ?c (call + 1.0 ?cl))  

MethodTrainCompetencyExamplesExercisesStop  
  (same ?cl 4.0)  
  (setCompetencyLevel ?c ?cl)  
  (illustrateCompetencySection ?competency ?c)  
  (practiceCompetencySection ?competency ?c)  
  (!deleteSetCompetencyLevel ?c ?cl)  

;; The following two operators serve to add manually information  
;; about the user in the world state. This is used is  
;; trainCompetency for selecting exercises and examples at given  
;; competency levels.  
(:operator (!!setCompetencyLevel ?c ?cl)  
  ; pre  
  ;(call PrintDebug Setting competency level ?c ?cl)  
  ; add  
  (learnerProperty hasCompetencyLevel ?c ?cl)  
  (set (learnerProperty hasCompetencyLevel ?c))  

(:operator (!!deleteSetCompetencyLevel ?c ?cl)  
  ; pre  
  ;(call PrintDebug Removing set competency level ?c ?cl)  
  ; delete  
  (learnerProperty hasCompetencyLevel ?c ?cl)
(learnerProperty hasCompetencyLevel ?c ?cl)
(set (learnerProperty hasCompetencyLevel ?c))
; add
()}

(:method (practiceCompetencySection ?competency ?c)
MethodPracticeCompetencySection
()
{
(!startSection Exercises (?c))
(trainWithSingleExercise ?c very_easy ?competency)
(trainWithSingleExercise ?c very_easy ?competency)
(trainWithSingleExercise ?c easy ?competency)
(trainWithSingleExercise ?c easy ?competency)
(trainWithSingleExercise ?c medium ?competency)
(trainWithSingleExercise ?c medium ?competency)
(trainWithSingleExercise ?c difficult ?competency)
(trainWithSingleExercise ?c difficult ?competency)
(trainWithSingleExercise ?c very_difficult ?competency)
(trainWithSingleExercise ?c very_difficult ?competency)
(!endSection)
)

(:method (illustrateCompetencySection ?competency ?c)
MethodIllustrateCompetencySection
()
{
(!startSection Examples (?c))
(illustrateWithSingleExample ?c very_easy ?competency)
(illustrateWithSingleExample ?c very_easy ?competency)
(illustrateWithSingleExample ?c easy ?competency)
(illustrateWithSingleExample ?c easy ?competency)
(illustrateWithSingleExample ?c medium ?competency)
(illustrateWithSingleExample ?c medium ?competency)
(illustrateWithSingleExample ?c difficult ?competency)
(illustrateWithSingleExample ?c difficult ?competency)
(illustrateWithSingleExample ?c very_difficult ?competency)
(illustrateWithSingleExample ?c very_difficult ?competency)
(!endSection)
)
)

; ; calculating prerequisites
; ; scenario GuidedTour
; ; This method calculates the prerequisites. The time variable
; ; helps to keep track of the concepts that were inserted in the
; ; current walk through the dependencies.
(:method (learnPrerequisitesConceptsGT ?c)
MethodLearnPrerequisitesConceptsGT
{
(collectUnknownPrereqIgnoreLearningContext ?c ?result)
(not (same ?result nil))
{
(learnPrerequisitesConceptsGT ?result)
}
MethodLearnPrerequisitesConceptsGTFallback
;; In case there are no prerequisites, do nothing.
()()

;; A helper method for calculating the prerequisites.
(:method (learnPrerequisitesConceptsGTH (?first . ?rest))
MethodLearnPrerequisitesConceptsGTHConcept()
;; insert it
(llearnSingleConceptGT ?first)
(llearnPrerequisitesConceptsGTH ?rest)
)

;; this is the case for the empty list. The above method does not
;; match for the empty list.
(:method (learnPrerequisitesConceptsGTH nil)
MethodLearnPrerequisitesConceptsGTHFallback()
)

;; calculating prerequisites
;; scenario LearnNew
(:method (learnPrerequisitesConceptsShortSection ?c)
MethodLearnPrerequisitesConceptsShort()

(learnPrerequisitesConceptsShortSection! ?c)
)

(:method (learnPrerequisitesConceptsShortSection ?c)
MethodLearnPrerequisitesConceptsShortNoPrerequisites()
;; In this case, there are no prerequisites to be
;; inserted. Therefore, we do nothing.
()
()
)

(:method (learnPrerequisitesConceptsShortSection! ?c)
MethodLearnPrerequisitesConceptsShortSection!

(collectUnknownPrereq ?c ?result)
;; (call PrintDebug Collected Prereq ?result)
(not (same ?result nil))

(startSection Prerequisites (?c))
text Prerequisites (?c)
(insertAllElements ?result)
(endSection)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:method (motivate ?c)
MethodMotivate()

(motivate! ?c)
)

;; don't fail in case the above can not be fulfilled
(:method (motivate ?c)
MethodMotivateFallback()
)

;; introduction in a proper section
(:method (introduceByIntroductionSection ?c)
MethodtIntroduceByIntroductionSection()
)
Deliverable D24

Tutorial Component

©LeActiveMath Consortium 2006
(:method (conclude ?c)
  MethodConcludeFallback
  ()
)

;; Used in LearnNew
(:method (problem ?c)
  MethodProblem
  ()
  (problem! ?c)
)

(:method (problem ?c)
  MethodProblemFallback
  ()
)

;; commented out till the mapping is done
;; (:method (problem! ?c)
;;   MethodProblem!
;;   (learnerProperty hasEducationalLevel ?el)
;;   (assignIterator ?element
;;     (call
;;       GetElements ((class Problem)
;;         (relation isFor ?c)
;;         (property hasLearningContext ?el)
;;       )))
;;   ; insert it
;;   (insertElementOnce! ?element)
;;   )
;;

;; Used in LearnNew
(:method (introductionExamplify ?c)
  MethodIntroductionExamplify
  ()
  (introductionExamplify! ?c)
)

(:method (introductionExamplify ?c)
  MethodIntroductionExamplifyFallback
  ()
)

;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; motivate

(:method (motivate! ?c)
  idealMethodMotivationByVeryEasyExercise
  ;; if the learner is not anxious, retrieve an introducing
  ;; exercise for the concept that matches the user’s
  ;; educational level
  (learnerProperty hasEducationalLevel ?el)
  (learnerProperty hasAnxiety ?c ?an)
  (call <= ?an 2)
  (assignIterator ?element
    (call
      GetElements ((class Exercise)
        (class Introduction)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasDifficulty very_easy)
      ))
    )
  )
  ; insert it
  (insertIfReady! ?element)
(:method (motivate! ?c)
  idealMethodMotivationByEasyExercise
  ;; if the learner is not anxious, retrieve an introducing
  ;; exercise for the concept that matches the user’s
  ;; educational level
  (assignIterator ?element
    (call
      GetElements ((class Exercise)
        (class Introduction)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasDifficulty easy))))
  ; insert it
  (insertIfReady! ?element)
)

(:method (motivate! ?c)
  idealMethodMotivationByVeryEasyExample
  ;; retrieve an introducing example for the concept that matches the
  ;; user’s educational level
  (assignIterator ?element
    (call
      GetElements ((class Example)
        (class Introduction)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasDifficulty very_easy))))
  ; insert it
  (insertIfReady! ?element ?c)
)

(:method (motivate! ?c)
  idealMethodMotivationByEasyExample
  ;; retrieve an introducing example for the concept that matches the
  ;; user’s educational level
  (assignIterator ?element
    (call
      GetElements ((class Example)
        (class Introduction)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasDifficulty easy))))
  ; insert it
  (insertIfReady! ?element ?c)
)

(:method (motivate! ?c)
  idealMethodMotivationByText
  ;; retrieve a text for the concept that matches the
  ;; user’s educational level
  (assignIterator ?element
    (call
      GetElements ((class Introduction)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasDifficulty easy)
        (property hasDifficulty very_easy))))
  ; insert it
  (insertIfReady! ?element ?c)
)
)))
)
; insert it
{
	(insertIfReady! ?element ?c)
}
)

:(method (motivate! ?c)
	fallbackMethodMotivationByMotivationLowerEducationalLevel
; retrieve a motivation for the concept that matches the
; user’s allowed educational level
{(learnerProperty hasAllowedEducationalLevel ?aels)
	(assignIterator ?el ?aels)
	(assignIterator ?element
	
call
	GetElements ((class Introduction)
		(relation isFor ?c)
		(property hasLearningContext ?el)
	))
}
; insert it
{
	(insertIfReady! ?element ?c)
}
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; introduce

:(method (introduceShort! ?c)
	MethodIntroduceEducationalLevel
; retrieve an introduction for the concept that matches
; the user’s educational level
{(learnerProperty hasEducationalLevel ?el)
	(assign ?elements (call
	GetElements ((class Introduction)
		(relation isFor ?c)
		(property hasLearningContext ?el)
	))
	(not (same ?elements nil))
		(assignIterator ?element ?elements)
}
; insert it
{
	(insertIfReady! ?element ?c)
}
)

MethodIntroduceEducationalLevelFallback
; retrieve an introduction for the concept that matches
; the user’s allowed educational level
{(learnerProperty hasAllowedEducationalLevel ?aels)
	(assignIterator ?el ?aels)
	(assign ?elements (call
	GetElements ((class Introduction)
		(relation isFor ?c)
		(property hasLearningContext ?el)
	))
	(not (same ?elements nil))
		(assignIterator ?element ?elements)
}
; insert it
{
	(insertIfReady! ?element ?c)
}
)

MethodIntroduceEducationalLevelFallbackAny
; retrieve any introduction for the concept
(assign ?elements (call
GetElements ((class Introduction)
		(relation isFor ?c)
	))
		(assignIterator ?element ?elements)
}
; insert it
{
	(insertIfReady! ?element ?c)
}
(:method (introduceShort! ?c)
  MethodIntroduceEducationalLevel
  ; retrieve an introduction for the concept that matches
  ; any the user’s allowed educational level
  (learnerProperty hasAllowedEducationalLevel ?aels)
  (assignIterator ?el ?aels)
  (assignIterator ?element
    (call
      GetElements ((class Introduction)
      (relation isFor ?c)
      (property hasLearningContext ?el)))
  )
  ; insert it
  (insertIfReady! ?element ?c)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; explain
(:method (explain! ?c)
  MethodExplain!Ideal
  ; retrieve an explanation for the concept that matches the
  ; user’s educational level
  (learnerProperty hasEducationalLevel ?el)
  (assignIterator ?element
    (call
      GetElements ((class Remark)
      (relation isFor ?c)
      (property hasLearningContext ?el)))
  )
  ; insert it
  (insertIfReady! ?element ?c)
)

MethodExplain!Fallback
; retrieve an explanation for the concept that matches any
; of the user’s allowed educational levels
(learnerProperty hasAllowedEducationalLevel ?aels)
(assignIterator ?el ?aels)
(assignIterator ?element
  (call
    GetElements ((class Remark)
    (relation isFor ?c)
    (property hasLearningContext ?el)))
)  ; insert it
  (insertIfReady! ?element ?c)
)

MethodExplain!FallbackAny
(); retrieve any explanation for the concept
(learnerProperty hasEducationalLevel ?el)
(assignIterator ?element
  (call
    GetElements ((class Remark)
    (relation isFor ?c)))
)  ; insert it
  (insertIfReady! ?element ?c)
)
;;; conclude

(:method (conclude! ?c)
  MethodConclude!Ideal
  ; retrieve a conclusion for the concept that matches the
  ; user's educationalLevel
  (learnerProperty hasEducationalLevel ?el)
  (assignIterator ?element
   (call
    GetElements ((class Conclusion)
                 (relation isFor ?c)
                 (property hasLearningContext ?el)))
  )
  ; insert it
  (insertIfReady! ?element ?c)
)

MethodConclude!Fallback
  ; retrieve a conclusion for the concept that matches any
  ; of the user's allowed educationalLevel
  (learnerProperty hasAllowedEducationalLevel ?aels)
  (assignIterator ?el ?aels)
  (assignIterator ?element
   (call
    GetElements ((class Conclusion)
                 (relation isFor ?c)
                 (property hasLearningContext ?el)))
  )
  ; insert it
  (insertIfReady! ?element ?c)
)

MethodConclude!FallbackAny
  ; retrieve any conclusion for the concept
  (assignIterator ?element
   (call
    GetElements ((class Conclusion)
                 (relation isFor ?c)))
  )
  ; insert it
  (insertIfReady! ?element ?c)

);

;;; examplify

(:method (introductionExamplify! ?c)
  MethodIntroductionExamplify!
  ()
  ;; We try to insert three examples.
  (insertIntroductionExample! ?c)
  (insertIntroductionExample! ?c)
  (insertIntroductionExample! ?c)
)

(:method (insertIntroductionExample ?c)
  MethodInsertIntroductionExample
  ()
)
((insertIntroductionExample! ?c))

(:method (insertIntroductionExample ?c)
  MethodInsertIntroductionExample!Fallback
  ()
  ()
)

(:method (insertIntroductionExample! ?c)
  MethodInsertVeryEasyIntroductionExample!
  (assignIterator ?element
    (call GetElements ((class Example) (class Introduction) (relation isFor ?c) (property hasDifficulty very_easy)))
  )
  ((insertElementIfReady! ?element ?c)
  )
)

(:method (insertIntroductionExample! ?c)
  MethodInsertEasyIntroductionExample!
  (assignIterator ?element
    (call GetElements ((class Example) (class Introduction) (relation isFor ?c) (property hasDifficulty easy)))
  )
  ((insertElementIfReady! ?element ?c)
  )
)

(:method (insertIntroductionExample! ?c)
  MethodInsertEasyIntroductionExample!
  (assignIterator ?element
    (call GetElements ((class Example) (class Introduction) (relation isFor ?c) (property hasDifficulty medium)))
  )
  ((insertElementIfReady! ?element ?c)
  )
)

;; Used in guided tour
(:method (illustrateWithIncreasedDiffSection ?c)
  MethodIllustrateWithIncreasedDiffSection
  (
    :only-proceed-if-there-are-any-examples-with-difficulty-values
    (or (call GetElements ((class Example) (relation isFor ?c) (property hasDifficulty very_easy)))
      (call GetElements ((class Example) (relation isFor ?c) (property hasDifficulty easy)))
      (call GetElements ((class Example) (relation isFor ?c) (property hasDifficulty medium)))
      (call GetElements ((class Example) (relation isFor ?c) (property hasDifficulty difficult)))
      (call GetElements ((class Example) (relation isFor ?c) (property hasDifficulty very_difficult)))
    )
  )
  (!startSection Examples (?c))
  (illustrateWithIncreasedDiff ?c)
  (!endSection)
)

MethodIllustrateWithIncreasedDiffSectionAnyExample
  (;; otherwise check whether there are any examples at all
   (getNonInserted ?elements (call GetElements ((class Example) (relation isFor ?c)))
   (not (same ?elements nil))
  )
)
(:method (illustrateWithIncreasedDiffSection ?c)
  MethodIllustrateWithIncreasedDiffSectionFallback
  ()
  ()
)

(:method (illustrateWithIncreasedDiff ?c)
  MethodIllustrateWithIncreasedDiff1
  (:method (illustrateWithIncreasedDiff 2)
    MethodIllustrateWithIncreasedDiff2
    (:method (illustrateWithIncreasedDiff 2)
      MethodIllustrateWithIncreasedDiff3
      (:method (insertExample 3)
        MethodInsertExample
        ()
        (insertExample! 3)
      )
      (:method (insertExample 1)
        MethodInsertExampleFallback
        ()
        ()
      )
    )
  )
)

;; don't fail in case the above can not be fulfilled
(:method (illustrateWithIncreasedDiffSection ?c)
  MethodIllustrateWithIncreasedDiffSectionFallback
  ()
  ()
)

;; don't fail in case the above can not be fulfilled
(:method (illustrateWithIncreasedDiffSection ?c)
  MethodIllustrateWithIncreasedDiffSectionFallback
  ()
  ()
)

;; don't fail in case the above can not be fulfilled
(:method (illustrateWithIncreasedDiffSection ?c)
  MethodIllustrateWithIncreasedDiffSectionFallback
  ()
  ()
)
(insertElementOnce ?element)
)

(:method (insertAllExamples ?c ?difficulty)
MethodInsertAllExamples
((assign ?elements
   (call
      GetElements ((class Example)
      (relation isFor ?c)
      (property hasDifficulty ?difficulty))))
  )
  (insertAllElements ?elements)
);

;; in case there are no examples
MethodInsertAllExamplesFallback
()
()

;; LeAM example selection
(:method (illustrateSection ?c)
MethodIllustrateSection
()
(!!startSection Examples (?c))
  (illustrate! ?c)
  (!!endSection)
)

(:method (illustrateSection ?c)
MethodIllustrateSectionFallback
()
)

;; select up to six examples
(:method (illustrate! ?c)
MethodIllustrate!
()
(!!illustrate! ?c)
)

(:method (illustrate ?c)
MethodIllustrate
()
(illustrate! ?c)
)

(:method (illustrate ?c)
MethodIllustrateFallback
()
)

(:method (illustrateH! ?c ?amount ?originalAmount)
MethodIllustrateH!Finished
;; all required examples were inserted, stop.
((call = ?amount 0))
()
)

(:method (illustrateH! ?c ?amount ?originalAmount)
MethodIllustrateH!Continue
;; if there are still examples to insert, continue
((call > ?amount 0))
(!!illustrateWithSingleExample! ?c)
(!!illustrateH! ?c (call - ?amount 1) ?originalAmount))

(:method (illustrateH! ?c ?amount ?originalAmount)
MethodIllustrateH!Fail
;; did we insert any examples? If not, fail
((call < ?amount ?originalAmount))
()}

(:method (illustrateNew! ?c)
MethodIllustrateNew
;; we first check whether there is a subtask that can be
;; fulfilled (the conditions correspond to the conditions
;; of the fallback method)
:first
(learnerProperty hasAllowedEducationalLevel ?aels)
(learnerProperty hasEducationalLevel ?edl)
(assignIterator ?el (?edl . ?aels))
(call
GetElements
((class Example)
(relation isFor ?c)
(property hasLearningContext ?el)
))
)
)

(::method (illustrateWithSingleExample ?c)
MethodIllustrateWithSingleExample
;; try to insert a suitable example
(:method (illustrateWithSingleExample! ?c)
MethodIllustrateWithSingleExampleDiffComp
())
(() ((illustrateWithSingleExample! ?c ?difficulty ?competence)))
)

(:method (illustrateWithSingleExample ?c ?difficulty ?competence)
MethodIllustrateWithSingleExampleDiffComp
;; did we insert any examples? If not, fail
((call < ?amount ?originalAmount))
()}

(c)LeActiveMath Consortium 2006
(method (illustrateWithSingleExample ! ?c)
  ;; best possible method: select example with matching
  ;; hasField and educational&competence level
  selectExampleFieldEdulevCompetencelev
  (call GetElements
    ((class Example)
     (relation isFor ?c)
     (property hasLearningContext ?el)
     (property hasCompetencyLevel ?ex_cl)
     (property hasField ?field)))
  ;;(assignIterator ?example ?examples)
  )
  (insertWithVariantsIfReady ! ?example !)
  )

(:method (illustrateWithSingleExample ! ?c)
  ;; best possible method: select example with matching
  ;; hasField and educational level
  selectExampleFieldEdulev
  (call GetElements
    ((class Example)
     (relation isFor ?c)
     (property hasLearningContext ?el)
     (property hasField ?field)))
  ;;(assignIterator ?example ?examples)
  )
  (insertWithVariantsIfReady ! ?example !)
  )

(:method (illustrateWithSingleExample ! ?c)
  ;; best possible method: select example with matching
  ;; educational&competence level
  selectExampleEdulevCompetencelevDiffComp
  (call GetElements
    ((class Example)
     (relation isFor ?c)
     (property hasLearningContext ?el)
     (property hasField ?field)))
  ;;(assignIterator ?example ?examples)
  )
  (insertWithVariantsIfReady ! ?example !)
  )

(:method (illustrateWithSingleExample ! ?c)
  ;; best possible method: select example with matching
  ;; educational&competence level
  selectExampleEdulevCompetencelevDiffComp
  (call GetElements
    ((class Example)
     (relation isFor ?c)
     (property hasLearningContext ?el)
     (property hasField ?field)))
  ;;(assignIterator ?example ?examples)
  )
  (insertWithVariantsIfReady ! ?example !)
  )

;; try to insert an appropriate example
;;; example selection

;; best possible method: select example with matching
;; hasField and educational&competence level
selectExampleFieldEdulevCompetencelev

;; best possible method: select example with matching
;; hasField and educational level
selectExampleFieldEdulev

;; best possible method: select example with matching
;; educational&competence level
selectExampleEdulevCompetencelevDiffComp

;; try to insert an appropriate example
;;; example selection

;; best possible method: select example with matching
;; hasField and educational&competence level
selectExampleFieldEdulevCompetencelev

;; best possible method: select example with matching
;; hasField and educational level
selectExampleFieldEdulev

;; best possible method: select example with matching
;; educational&competence level
selectExampleEdulevCompetencelevDiffComp
(learnerProperty hasCompetencyLevel ?c ?cl)
(equivalent ?cl ?ex_cl)
/removeNotReady ?examples ?c
(assignIterator ?example
(call
GetElements
((class Example)
(relation isFor ?c)
(property hasLearningContext ?el)
(property hasCompetencyLevel ?ex_cl))
);
(assignIterator ?example ?examples)
)
(insertWithVariantsIfReady! ?example ?c)
;;(insertElementOnce! ?example)
)
(:method (illustrateWithSingleExample! ?c)
;; fallback method: select example with matching
;; hasField and competence level, lower educational-level
selectExampleFieldLowerEducationalLevelCompetenceLevelDiffComp
(
(learnerProperty hasField ?field)
(learnerProperty hasAllowedEducationalLevel ?aels)
(assignIterator ?el ?aels)
(learnerProperty hasCompetencyLevel ?c ?cl)
(equivalent ?cl ?ex_cl)
)
/removeNotReady ?examples ?c
(assignIterator ?example
(call
GetElements
((class Example)
(relation isFor ?c)
(property hasLearningContext ?el)
(property hasCompetencyLevel ?ex_cl)
(property hasField ?field))
);
(assignIterator ?example ?examples)
)
(insertWithVariantsIfReady! ?example ?c)
;;(insertElementOnce! ?example)
)
(:method (illustrateWithSingleExample! ?c)
;; fallback method: select example with matching
;; hasField and lower educational level
selectExampleFieldLowerEducationalLevelDiffComp
(
(learnerProperty hasField ?field)
(learnerProperty hasAllowedEducationalLevel ?aels)
(assignIterator ?el ?aels)
)
/removeNotReady ?examples ?c
(assignIterator ?example
(call
GetElements
((class Example)
(relation isFor ?c)
(property hasLearningContext ?el)
(property hasField ?field))
);
(assignIterator ?example ?examples)
)
(insertWithVariantsIfReady! ?example ?c)
;;(insertElementOnce! ?example)
)
(:method (illustrateWithSingleExample! ?c)
;; fallback method: select example with matching
;; competence level and lower educational level
selectExampleEducationalLevelCompetenceLevelDiffComp
(
(learnerProperty hasAllowedEducationalLevel ?aels)
(assignIterator ?el ?aels)
)
(learnerProperty hasCompetencyLevel ?c ?cl)
  (equivalent ?cl ?ex_cl)
  (removeNotReady ?examples ?c)
  (assignIterator ?example)
  (call GetElements
    ((class Example)
     (relation isFor ?c)
     (property hasLearningContext ?el)
     (property hasCompetencyLevel ?ex_cl)
     )))
  (assignIterator ?example ?examples)
  )
  (insertWithVariantsIfReady! ?example ?c)
  (insertElementOnce! ?example)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; examples with difficulty and competencies
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(:method (illustrateWithSingleExample! ?c ?difficulty ?competency)
  ;; best possible method: select example with matching hasField and educational&competence level
  selectExampleFieldEdulevCompetencelevDiffComp
  {
    (learnerProperty hasField ?field)
    (learnerProperty hasEducationalLevel ?el)
    (learnerProperty hasCompetencyLevel ?c ?cl)
    (equivalent ?cl ?ex_cl)
    (removeNotReady ?examples ?c)
    (assignIterator ?example)
    (call GetElements
      ((class Example)
       (relation isFor ?c)
       (property hasLearningContext ?el)
       (property hasCompetencyLevel ?ex_cl)
       (property hasField ?field)
       (property hasDifficulty ?difficulty)
       (property hasCompetency ?competency)
       ))
      (assignIterator ?example ?examples)
      )
      (insertWithVariantsIfReady! ?example ?c)
      (insertElementOnce! ?example)
      )
    )

(:method (illustrateWithSingleExample! ?c ?difficulty ?competency)
  ;; best possible method: select example with matching hasField and educational level
  selectExampleFieldEdulevDiffComp
  {
    (learnerProperty hasField ?field)
    (learnerProperty hasEducationalLevel ?el)
    (removeNotReady ?examples ?c)
    (assignIterator ?example)
    (call GetElements
      ((class Example)
       (relation isFor ?c)
       (property hasLearningContext ?el)
       (property hasField ?field)
       (property hasDifficulty ?difficulty)
       (property hasCompetency ?competency)
       )))
      (assignIterator ?example ?examples)
      )
      (insertWithVariantsIfReady! ?example ?c)
      (insertElementOnce! ?example)
      )
    )

©LeActiveMath Consortium 2006
(:method (illustrateWithSingleExample! ?c ?difficulty ?competency)
 ;; best possible method: select example with matching educational&competence level
 selectExampleEdulevCompetencelevDiffComp
  (learnerProperty hasEducationalLevel ?el)
  (learnerProperty hasCompetencyLevel ?c ?cl)
  (equivalent ?cl ?ex_cl)
  (removeNotReady ?examples ?c)
  (assignIterator ?example)
  (call GetElements
    ((class Example)
      (relation isFor ?c)
      (property hasLearningContext ?el)
      (property hasCompetencyLevel ?ex_cl)
      (property hasDifficulty ?difficulty)
      (property hasCompetency ?competency)
    )))
  (assignIterator ?example ?examples)
  (insertWithVariantsIfReady! ?example ?c)
  (insertElementOnce! ?example)
)

(:method (illustrateWithSingleExample! ?c ?difficulty ?competency)
 ;; fallback method: select example with matching hasField and competence level, lower educational-level
 selectExampleFieldLowerEdulevCompetencelevDiffComp
  (learnerProperty hasField ?field)
  (learnerProperty hasAllowedEducationalLevel ?aels)
  (assignIterator ?el ?aels)
  (learnerProperty hasCompetencyLevel ?c ?cl)
  (equivalent ?cl ?ex_cl)
  (removeNotReady ?examples ?c)
  (assignIterator ?example)
  (call GetElements
    ((class Example)
      (relation isFor ?c)
      (property hasLearningContext ?el)
      (property hasCompetencyLevel ?ex_cl)
      (property hasDifficulty ?difficulty)
      (property hasCompetency ?competency)
    )))
  (assignIterator ?example ?examples)
  (insertWithVariantsIfReady! ?example ?c)
  (insertElementOnce! ?example)
)

(:method (illustrateWithSingleExample! ?c ?difficulty ?competency)
 ;; fallback method: select example with matching hasField and lower educational level
 selectExampleFieldLowerEdulevDiffComp
  (learnerProperty hasField ?field)
  (learnerProperty hasAllowedEducationalLevel ?aels)
  (assignIterator ?el ?aels)
  (call GetElements
    ((class Example)
      (relation isFor ?c)
      (property hasLearningContext ?el)
      (property hasField ?field)
      (property hasDifficulty ?difficulty)
      (property hasCompetency ?competency)
    )))
  (assignIterator ?example ?examples)
)
(insertWithVariantsIfReady! ?example ?c)
  ;; (insertElementOnce! ?example)
)
)

(:method (illustrateWithSingleExample! ?c ?difficulty ?competency)
  ;; fallback method: select example with matching
  ;; competence level and lower educational level
  selectExampleEdulevCompetencelevDiffComp
  {
    (learnerProperty hasAllowedEducationalLevel ?aels) (assignIterator ?el ?aels)
    (learnerProperty hasCompetencyLevel ?c ?cl)
    (equivalent ?cl ?ex_cl)
    ;; (removeNotReady ?examples ?c)
    (assignIterator ?example)
    (call GetElements ((class Example) (relation isFor ?c) (property hasLearningContext ?el) (property hasCompetencyLevel ?ex_cl) (property hasDifficulty ?difficulty) (property hasCompetency ?competency)))
  ;; (assignIterator ?example ?examples)
  )
  (insertWithVariantsIfReady! ?example ?c)
  ;; (insertElementOnce! ?example)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; exercise
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(:method (trainWithIncreasedDiffSection ?c)
  MethodTrainWithIncreasedDiffSection
  {
    ;; only proceed if there are any exercises with difficulty values
    ;; (call GetElements ((class Exercise) (relation isFor ?c) (property hasDifficulty very_easy))
    (call GetElements ((class Exercise) (relation isFor ?c) (property hasDifficulty easy)))
    (call GetElements ((class Exercise) (relation isFor ?c) (property hasDifficulty medium)))
    (call GetElements ((class Exercise) (relation isFor ?c) (property hasDifficulty difficult)))
    (call GetElements ((class Exercise) (relation isFor ?c) (property hasDifficulty very_difficult)))
  }))
  (startSection Exercises (?c))
  (trainWithIncreasedDiff ?c)
  (endSection)
)

MethodIllustrateWithIncreasedDiffSectionAnyExercises
{
  ;; otherwise check whether there are any exercises at all
  (getNonInserted ?elements (call GetElements ((class Exercise) (relation isFor ?c))))
  (not (same ?elements nil))
}

  (startSection Exercises (?c))
  (insertUpTo 10 ?elements)
  (endSection)
)

;; don't fail in case the above can not be fulfilled
(:method (trainWithIncreasedDiffSection ?c)
  MethodTrainWithIncreasedDiffSectionFallback
  ()
  ()
)

(:method (trainWithIncreasedDiff ?c)
  MethodTrainWithIncreasedDiff1
  {
    (learnerProperty hasCompetencyLevel ?c ?competencyLevel)
    (call < ?competencyLevel 2)
  }
(insertElementsOfType ?c Exercise very_easy 4)
(insertElementsOfType ?c Exercise easy 3)
(insertElementsOfType ?c Exercise medium 2)
(insertElementsOfType ?c Exercise difficult 1)
(insertElementsOfType ?c Exercise very_difficult 1)
)

MethodTrainWithIncreasedDiff2
(
(learnerProperty hasCompetencyLevel ?c ?competencyLevel)
(call >= ?competencyLevel 2)
(call < ?competencyLevel 4)
)

(insertElementsOfType ?c Exercise very_easy 2)
(insertElementsOfType ?c Exercise easy 3)
(insertElementsOfType ?c Exercise medium 3)
(insertElementsOfType ?c Exercise difficult 2)
(insertElementsOfType ?c Exercise very_difficult 2)
)

MethodTrainWithIncreasedDiff3
(
(learnerProperty hasCompetencyLevel ?c ?competencyLevel)
(call >= ?competencyLevel 4)
)

(insertElementsOfType ?c Exercise very_easy 1)
(insertElementsOfType ?c Exercise easy 1)
(insertElementsOfType ?c Exercise medium 2)
(insertElementsOfType ?c Exercise difficult 4)
(insertElementsOfType ?c Exercise very_difficult 4)
)

(:method (insertExercise ?c ?difficulty)
  MethodInsertExcOfDifficulty
  ()
  ((insertExercise! ?c ?difficulty)))
)

(:method (insertExercise ?c ?difficulty)
  MethodInsertExcOfDifficultyFallback
  ()
  ()
)

(:method (insertExercise! ?c ?difficulty)
  MethodInsertExcOfDifficultyCritical
  ((assignIterator ?element
    (call
      GetElements
      ((class Exercise)
        (relation isFor ?c)
        (property hasDifficulty ?difficulty)))))
  )

  (insertElementOnce! ?element)
  )
)

(:method (insertAllExercises ?c ?difficulty)
  MethodInsertAllExcOfDifficulty
  ((assign ?elements
    (call
      GetElements
      ((class Exercise)
        (relation isFor ?c)
        (property hasDifficulty ?difficulty)))))
  )

  (insertAllElements ?elements)
  )

()
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; introducing exercises

(:method (insertIntroductionExercise ?c)
 MethodInsertIntroductionExercise
 ()
 ((insertIntroductionExercise! ?c))
)

(:method (insertIntroductionExercise ?c)
 MethodInsertIntroductionExercise!Fallback
 ()
 ()
)

(:method (insertIntroductionExercise! ?c)
 MethodInsertVeryEasyIntroductionExercise!
 (assignIterator ?element
 (call
 GetElements ((class Exercise)
 (class Introduction)
 (relation isFor ?c)
 (property hasDifficulty very_easy)
 )))
)

 (:method (insertIntroductionExercise! ?c)
 MethodInsertEasyIntroductionExercise!
 (assignIterator ?element
 (call
 GetElements ((class Exercise)
 (class Introduction)
 (relation isFor ?c)
 (property hasDifficulty easy)
 )))
)

 (:method (insertIntroductionExercise! ?c)
 MethodInsertEasyIntroductionExercise!
 (assignIterator ?element
 (call
 GetElements ((class Exercise)
 (class Introduction)
 (relation isFor ?c)
 (property hasDifficulty medium)
 )))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; exercise selection

;; try to insert an appropriate exercise
 (:method (trainWithSingleExercise ?c)
 MethodTrainWithSingleExercise
 ()
 ((trainWithSingleExercise! ?c))
)

 (:method (trainWithSingleExercise ?c)
 MethodTrainWithSingleExerciseFallback
 ()
)
(:method (train! ?c)
  MethodTrain!
)

(:method (trainWithSingleExercise ! ?c)
  MethodTrainWithSingleExercise

  (trainWithSingleExercise ?c very_easy think)
  (trainWithSingleExercise ?c very_easy solve)
  (trainWithSingleExercise ?c very_easy represent)
  (trainWithSingleExercise ?c very_easy language)
  (trainWithSingleExercise ?c very_easy argue)
  (trainWithSingleExercise ?c very_easy tools)
  (trainWithSingleExercise ?c very_easy communicate)

  (trainWithSingleExercise ?c easy think)
  (trainWithSingleExercise ?c easy solve)
  (trainWithSingleExercise ?c easy represent)
  (trainWithSingleExercise ?c easy language)
  (trainWithSingleExercise ?c easy model)
  (trainWithSingleExercise ?c easy argue)
  (trainWithSingleExercise ?c easy tools)
  (trainWithSingleExercise ?c easy communicate)

  (trainWithSingleExercise ?c medium think)
  (trainWithSingleExercise ?c medium solve)
  (trainWithSingleExercise ?c medium represent)
  (trainWithSingleExercise ?c medium language)
  (trainWithSingleExercise ?c medium model)
  (trainWithSingleExercise ?c medium argue)
  (trainWithSingleExercise ?c medium tools)
  (trainWithSingleExercise ?c medium communicate)

  (trainWithSingleExercise ?c difficult think)
  (trainWithSingleExercise ?c difficult solve)
  (trainWithSingleExercise ?c difficult represent)
  (trainWithSingleExercise ?c difficult language)
  (trainWithSingleExercise ?c difficult model)
  (trainWithSingleExercise ?c difficult argue)
  (trainWithSingleExercise ?c difficult tools)
  (trainWithSingleExercise ?c difficult communicate)

  (trainWithSingleExercise ?c very_difficult think)
  (trainWithSingleExercise ?c very_difficult solve)
  (trainWithSingleExercise ?c very_difficult represent)
  (trainWithSingleExercise ?c very_difficult language)
  (trainWithSingleExercise ?c very_difficult model)
  (trainWithSingleExercise ?c very_difficult argue)
  (trainWithSingleExercise ?c very_difficult tools)
  (trainWithSingleExercise ?c very_difficult communicate)
)

(:method (trainWithSingleExercise ?c)
  MethodTrainWithSingleExercise

  (trainWithSingleExercise ?c very_easy think)
  (trainWithSingleExercise ?c very_easy solve)
  (trainWithSingleExercise ?c very_easy represent)
  (trainWithSingleExercise ?c very_easy language)
  (trainWithSingleExercise ?c very_easy argue)
  (trainWithSingleExercise ?c very_easy tools)
  (trainWithSingleExercise ?c very_easy communicate)

  (trainWithSingleExercise ?c easy think)
  (trainWithSingleExercise ?c easy solve)
  (trainWithSingleExercise ?c easy represent)
  (trainWithSingleExercise ?c easy language)
  (trainWithSingleExercise ?c easy model)
  (trainWithSingleExercise ?c easy argue)
  (trainWithSingleExercise ?c easy tools)
  (trainWithSingleExercise ?c easy communicate)

  (trainWithSingleExercise ?c medium think)
  (trainWithSingleExercise ?c medium solve)
  (trainWithSingleExercise ?c medium represent)
  (trainWithSingleExercise ?c medium language)
  (trainWithSingleExercise ?c medium model)
  (trainWithSingleExercise ?c medium argue)
  (trainWithSingleExercise ?c medium tools)
  (trainWithSingleExercise ?c medium communicate)

  (trainWithSingleExercise ?c difficult think)
  (trainWithSingleExercise ?c difficult solve)
  (trainWithSingleExercise ?c difficult represent)
  (trainWithSingleExercise ?c difficult language)
  (trainWithSingleExercise ?c difficult model)
  (trainWithSingleExercise ?c difficult argue)
  (trainWithSingleExercise ?c difficult tools)
  (trainWithSingleExercise ?c difficult communicate)

  (trainWithSingleExercise ?c very_difficult think)
  (trainWithSingleExercise ?c very_difficult solve)
  (trainWithSingleExercise ?c very_difficult represent)
  (trainWithSingleExercise ?c very_difficult language)
  (trainWithSingleExercise ?c very_difficult model)
  (trainWithSingleExercise ?c very_difficult argue)
  (trainWithSingleExercise ?c very_difficult tools)
  (trainWithSingleExercise ?c very_difficult communicate)
)

;: best possible method: if motivation is high, then select
;: exercise with higher competency level
MethodSelectExerciseHighMotivation

  (LearnerProperty hasField ?field)
  (LearnerProperty hasEducationalLevel ?el)
  (LearnerProperty hasMotivation % ?m)
  (call >= ?m 4)
  (LearnerProperty hasCompetencyLevel % ?c1)
  (equivariant (call + 1 ?c1) ?ex_cl)
  ;(removeNotReady 'exercise ?c
  (assignIterator 'exercise ?c
  (call
    GetElements
    (class Exercise)
    (relation isFor ?c)
    (property hasLearningContext ?el)
    (property hasCompetencyLevel ?ex_cl)
    (property hasField ?field)
  )
  )
  ;(assignIterator 'exercise 'exercises)
  )
(:method (trainWithSingleExercise! ?c)
   ;; best possible method: if motivation is low, select
   ;; exercise on lower competency level.
   MethodSelectExerciseLowMotivation
   (
     (learnerProperty hasField ?field)
     (learnerProperty hasEducationalLevel ?el)
     (learnerProperty hasMotivation ?c ?m)
     (call <= ?m 1)
     (learnerProperty hasCompetencyLevel ?c ?cl)
     (equivalent (call - ?cl 1) ?ex_cl)
     (removeNotReady ?exercises ?c)
     (assignIterator ?exercise)
     (call GetElements
       ((class Exercise)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasCompetencyLevel ?ex_cl)
        (property hasField ?field))
     )
     (assignIterator ?exercise ?exercises)
   )
   ;; exercise on adequate competency level.
   MethodSelectExerciseAdequateCompetencyLevel
   (
     (learnerProperty hasField ?field)
     (learnerProperty hasEducationalLevel ?el)
     (learnerProperty hasCompetencyLevel ?c ?cl)
     (equivalent ?cl ?ex_cl)
     (removeNotReady ?exercises ?c)
     (assignIterator ?exercise)
     (call GetElements
       ((class Exercise)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasCompetencyLevel ?ex_cl)
        (property hasField ?field))
     )
     (assignIterator ?exercise ?exercises)
   )
   ;; select exercise from any field but correct learning context.
   MethodSelectExerciseAnyHasFieldCorrectEdLev
   (
     (learnerProperty hasEducationalLevel ?el)
     (learnerProperty hasCompetencyLevel ?c ?cl)
     (equivalent ?cl ?ex_cl)
     (removeNotReady ?exercises ?c)
     (assignIterator ?exercise)
     (call GetElements
       ((class Exercise)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasCompetencyLevel ?ex_cl))
     )
     (assignIterator ?exercise ?exercises)
   )
(:method (trainWithSingleExercise! ?c)
  ;; fallback method: select exercise on lower competency
  ;; level.
  MethodSelectExerciseLowerCompetencyLevel
  {
    (learnerProperty hasField ?field)
    (learnerProperty hasEducationalLevel ?el)
    (learnerProperty hasCompetencyLevel ?c ?cl)
    (equivalent (call - ?cl 1) ?ex_cl)
    (removeNotReady ?exercises ?c)
    (assignIterator ?exercise
      (call GetElements
        ((class Exercise)
          (relation isFor ?c)
          (property hasLearningContext ?el)
          (property hasCompetencyLevel ?ex_cl)
          (property hasField ?field)
          )))
    (assignIterator ?exercise ?exercises)
    )
    (insertWithVariantsIfReady! ?exercise ?c)
    (insertElementOnce! ?exercise)
  }
)

(:method (trainWithSingleExercise! ?c)
  ;; fallback method: select exercise from any field and
  ;; lower learning context.
  MethodSelectExerciseAnyHasFieldLowerCompetencyLevel
  {
    (learnerProperty hasEducationalLevel ?el)
    (learnerProperty hasCompetencyLevel ?c ?cl)
    (equivalent (call - ?cl 1) ?ex_cl)
    (removeNotReady ?exercises ?c)
    (assignIterator ?exercise
      (call GetElements
        ((class Exercise)
          (relation isFor ?c)
          (property hasLearningContext ?el)
          (property hasCompetencyLevel ?ex_cl)
          )))
    (assignIterator ?exercise ?exercises)
    )
    (insertWithVariantsIfReady! ?exercise ?c)
    (insertElementOnce! ?exercise)
  }
)

(:method (trainWithSingleExercise! ?c)
  ;; fallback method: if motivation is high but no
  ;; exercise with matching learning context was found,
  ;; select "lower" learning context.
  MethodSelectExerciseHighMotivationLowerEducationalLevel
  {
    (learnerProperty hasField ?field)
    (learnerProperty hasAllowedEducationalLevel ?aels)
    (assignIterator ?el ?aels)
    (learnerProperty hasMotivation ?c ?m)
    (call >= ?m 4)
    (learnerProperty hasCompetencyLevel ?c ?cl)
    (equivalent (call + 1 ?cl) ?ex_cl)
    (removeNotReady ?exercises ?c)
    (assignIterator ?exercise
      (call GetElements
        ((class Exercise)
          (relation isFor ?c)
          )))
  }
)

(assignIterator ?el ?aels)
(learnerProperty hasCompetencyLevel ?c ?cl)
(equivalent ?cl ?ex_cl)
;(removeNotReady ?exercises ?c)
(assignIterator ?exercise
(call
GetElements ((class Exercise)
(relation isFor ?c)
@property hasLearningContext ?el)
(property hasCompetencyLevel ?ex_cl)
)));
(assignIterator ?exercise ?exercises)
)
{
(insertWithVariantsIfReady! ?exercise ?c)
;(insertElementOnce! ?exercise)
}

(:method (trainWithSingleExercise! ?c)
;; fallback method: if no exercise with matching
;; learning context was found, select "lower" learning
;; context and lower competency level
MethodSelectExerciseLowerLCLowerCompLev
{
(learnerProperty hasField ?field)
(learnerProperty hasAllowedEducationalLevel ?aels)
(assignIterator ?el ?aels)
(learnerProperty hasCompetencyLevel ?c ?cl)
(equivalent (call - ?cl 1) ?ex_cl)
;(removeNotReady ?exercises ?c)
(assignIterator ?exercise
(call
GetElements ((class Exercise)
(relation isFor ?c)
@property hasLearningContext ?el)
(property hasCompetencyLevel ?ex_cl)
(property hasField ?field)
)));
(assignIterator ?exercise ?exercises)
)
{
(insertWithVariantsIfReady! ?exercise ?c)
;(insertElementOnce! ?exercise)
}

(:method (trainWithSingleExercise! ?c)
;; fallback method: select exercise from any field and any
;; allowed learning context and lower competency level
MethodSelectExerciseAnyFieldLowerLCLowerCompLev
{
(learnerProperty hasAllowedEducationalLevel ?aels)
(assignIterator ?el ?aels)
(learnerProperty hasCompetencyLevel ?c ?cl)
(equivalent (call - ?cl 1) ?ex_cl)
;(removeNotReady ?exercises ?c)
;(insertWithVariantsIfReady! ?exercise ?c)
;;(insertElementOnce! ?exercise)
})
;; same methods as above, extended with difficulty and competencies
;; __________________________________

(:method (trainWithSingleExercise ?c ?difficulty ?competency)
MethodTrainWithSingleExerciseDiffComp
()
)

(:method (trainWithSingleExercise ?c ?difficulty ?competency)
MethodTrainWithSingleExerciseDiffCompFallback
()
()
)

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
;; best possible method: if motivation is high, then select
;; exercise with higher competency level
MethodSelectExerciseHighMotivationDiffComp
{
  (learnerProperty hasField ?field)
  (learnerProperty hasEducationalLevel ?el)
  (learnerProperty hasMotivation ?c ?m)
  (call >= ?m 4)
  (learnerProperty hasCompetencyLevel ?c ?cl)
  (equivalent (call + 1 ?cl) ?ex_cl)
  (removeNotReady ?exercises ?c)
  (assignIterator ?exercise ?exercises)
  (call
    GetElements
    ((class Exercise)
      (relation isFor ?c)
      (property hasLearningContext ?el)
      (property hasCompetencyLevel ?ex_cl)
      (property hasField ?field)
      (property hasDifficulty ?difficulty)
      (property hasCompetency ?competency)
    )))
  (assignIterator ?exercise ?exercises)
  (insertWithVariantsIfReady! ?exercise ?c)
  (insertElementOnce! ?exercise)
)
)

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
;; best possible method: if motivation is low, select
;; exercise on lower competency level.
MethodSelectExerciseLowMotivationDiffComp
{
  (learnerProperty hasField ?field)
  (learnerProperty hasEducationalLevel ?el)
  (learnerProperty hasMotivation ?c ?m)
  (call <= ?m 1)
  (learnerProperty hasCompetencyLevel ?c ?cl)
  (equivalent (call - ?cl 1) ?ex_cl)
  (removeNotReady ?exercises ?c)
  (assignIterator ?exercise)
  (call
    GetElements
    ((class Exercise)
      (relation isFor ?c)
      (property hasLearningContext ?el)
      (property hasCompetencyLevel ?ex_cl)
      (property hasField ?field)
      (property hasDifficulty ?difficulty)
      (property hasCompetency ?competency)
    )))
  (assignIterator ?exercise ?exercises)
  (insertWithVariantsIfReady! ?exercise ?c)
  (insertElementOnce! ?exercise)
)
)
(:method (trainWithSingleExercise! ?c ?difficulty ?competency)

;; best possible method: select exercise on adequate competency
;; level.
MethodSelectExerciseAdequateCompetencyLevelDiffComp

{(learnerProperty hasField ?c
(learnerProperty hasEducationalLevel ?el)
(learnerProperty hasCompetencyLevel ?c ?cl)
(equivalent ?cl ?ex_cl)
; (removeNotReady ?exercises ?c
(assignIterator ?exercise
; (call
GetElements
((class Exercise)
(relation isFor ?c)
(property hasLearningContext ?el)
(property hasCompetencyLevel ?ex_cl)
(property hasDifficulty ?difficulty)
(property hasCompetency ?competency)
))))
; (assignIterator ?exercise ?exercises)
)
;
(insertWithVariantsIfReady! ?exercise ?c)
;; (insertElementOnce! ?exercise)
)
)

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)

;; best possible method: select exercise from any field but
;; correct learning context.
MethodSelectExerciseAnyHasFieldCorrectEdLevDiffComp

{(learnerProperty hasEducationalLevel ?el)
(learnerProperty hasCompetencyLevel ?c ?cl)
(equivalent ?cl ?ex_cl)
; (removeNotReady ?exercises ?c
(assignIterator ?exercise
; (call
GetElements
((class Exercise)
(relation isFor ?c)
(property hasLearningContext ?el)
(property hasCompetencyLevel ?ex_cl)
(property hasDifficulty ?difficulty)
(property hasCompetency ?competency)
))))
; (assignIterator ?exercise ?exercises)
)
;
(insertWithVariantsIfReady! ?exercise ?c)
;; (insertElementOnce! ?exercise)
)
)

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)

;; fallback method: select exercise on lower competency
;; level.
MethodSelectExerciseLowerCompetencyLevelDiffComp

{(learnerProperty hasField ?c
(learnerProperty hasEducationalLevel ?el)
(learnerProperty hasCompetencyLevel ?c ?cl)
(equivalent (call - ?cl 1) ?ex_cl)
; (removeNotReady ?exercises ?c
(assignIterator ?exercise
; (call
GetElements
((class Exercise)
(relation isFor ?c)
(property hasLearningContext ?el)
(property hasCompetencyLevel ?ex_cl)
(property hasDifficulty ?difficulty)
(property hasField ?field)
(property hasCompetency ?competency)
))))
; (assignIterator ?exercise ?exercises)
)
)
(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
  ;; fallback method: select exercise from any field and
  ;; lower learning context.
  MethodSelectExerciseAnyHasFieldLoverCompetencyLevelDiffComp
  
  (LearnerProperty hasField ?field)
  (LearnerProperty hasAllowedEducationalLevel ?aels)
  (assignIterator ?el ?aels)
  (LearnerProperty hasMotivation ?c ?m)
  (call <= ?m 1)
  (LearnerProperty hasCompetencyLevel ?c ?cl)
  (assignIterator ?exercise
    (call GetElements
      ((class Exercise)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasCompetencyLevel ?ex_cl)
        (property hasDifficulty ?difficulty)
        (property hasCompetency ?competency)
        )))
  )
  (insertWithVariantsIfReady! ?exercise ?c)
  (insertElementOnce! ?exercise)
)
)

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
  ;; fallback method: if motivation is low but no
  ;; exercise with matching learning context was found,
  ;; select "lower" learning context.
  MethodSelectExerciseLowMotivationLoverEducationalLevelDiffComp
  
  (LearnerProperty hasField ?field)
  (LearnerProperty hasAllowedEducationalLevel ?aels)
  (assignIterator ?el ?aels)
  (LearnerProperty hasField ?field)
  (LearnerProperty hasAllowedEducationalLevel ?aels)
  (assignIterator ?el ?aels)
  (LearnerProperty hasMotivation ?c ?m)
  (call <= ?m 1)
  (assignIterator ?exercise
    (call GetElements
      ((class Exercise)
        (relation isFor ?c)
        (property hasLearningContext ?el)
        (property hasCompetencyLevel ?ex_cl)
        (property hasField ?field)
        (property hasDifficulty ?difficulty)
        (property hasCompetency ?competency)
        )))
  )
  (insertWithVariantsIfReady! ?exercise ?c)
  (insertElementOnce! ?exercise)
)
)
(learnerProperty hasCompetencyLevel ?c ?cl)
(called (call - ?cl 1) ?ex_cl)
; (removeNotReady *exercises ?c
(assignIterator *exercise
(call GetElements
((class Exercise)
 (relation isFor ?c)
 (property hasLearningContext ?el)
 (property hasCompetencyLevel ?ex_cl)
 (property hasField ?field)
 (property hasDifficulty ?difficulty)
 (property hasCompetency ?competency)
))

; (assignIterator *exercise *exercises)
)

(insertWithVariantsIfReady! ?exercise ?c)
; ; (insertElementOnce! ?exercise)
)

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
 ;; fallback method: if no exercise with matching
 ;; learning context was found, select "lower" learning
 ;; context.
 MethodSelectExerciseLowerEducationalLevelDiffComp
 (learnerProperty hasField ?field)
 (learnerProperty hasAllowedEducationalLevel ?aels)
 (assignIterator ?el ?aels)
 (learnerProperty hasCompetencyLevel ?c ?cl)
 (equivalent ?cl ?ex_cl)
 ; (removeNotReady *exercises ?c
 (assignIterator *exercise
 (call GetElements
 ((class Exercise)
 (relation isFor ?c)
 (property hasLearningContext ?el)
 (property hasCompetencyLevel ?ex_cl)
 (property hasField ?field)
 (property hasDifficulty ?difficulty)
 (property hasCompetency ?competency)
 ))

; (assignIterator *exercise *exercises)
)

(insertWithVariantsIfReady! ?exercise ?c)
; ; (insertElementOnce! ?exercise)
)

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
 ;; fallback method: select exercise from any field and any
 ;; allowed learning context.
 MethodSelectExerciseAnyHasFieldDiffComp
 (learnerProperty hasAllowedEducationalLevel ?aels)
 (assignIterator ?el ?aels)
 (learnerProperty hasCompetencyLevel ?c ?cl)
 (equivalent ?cl ?ex_cl)
 ; (removeNotReady *exercises ?c
 (assignIterator *exercise
 (call GetElements
 ((class Exercise)
 (relation isFor ?c)
 (property hasLearningContext ?el)
 (property hasCompetencyLevel ?ex_cl)
 (property hasDifficulty ?difficulty)
 (property hasCompetency ?competency)
 ))

; (assignIterator *exercise *exercises)
)

(insertWithVariantsIfReady! ?exercise ?c)
; ; (insertElementOnce! ?exercise)
)
(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
  ;; fallback method: if no exercise with matching
  ;; learning context was found, select "lower" learning
  ;; context and lower competency level
  MethodSelectExerciseLowerLCLowerCompLevDiffComp
  
  (LearnerProperty hasField ?field)
  (LearnerProperty hasAllowedEducationalLevel ?aels)
  (assignIterator ?el ?aels)
  (LearnerProperty hasCompetencyLevel ?c ?cl)
  (equivalent (call - ?cl 1) ?ex_cl)
  ;;removeNotReady ?exercises ?c
  (assignIterator ?exercise ?exercises)
  (call GetElements
    (class Exercise)
    (relation isFor ?c)
    (property hasLearningContext ?el)
    (property hasCompetencyLevel ?ex_cl)
    (property hasDifficulty ?difficulty)
    (property hasCompetency ?competency)
    ))
  (assignIterator ?exercise ?exercises)
  (insertWithVariantsIfReady! ?exercise ?c)
  (insertElementOnce! ?exercise)
)

(:method (trainWithSingleExercise! ?c ?difficulty ?competency)
  ;; fallback method: select exercise from any field and any
  ;; allowed learning context and lower competency level
  MethodSelectExerciseAnyFieldLowerLCLowerCompLevDiffComp
  
  (LearnerProperty hasAllowedEducationalLevel ?aels)
  (assignIterator ?el ?aels)
  (LearnerProperty hasCompetencyLevel ?c ?cl)
  (equivalent (call - ?cl 1) ?ex_cl)
  ;;removeNotReady ?exercises ?c
  (assignIterator ?exercise ?exercises)
  (call GetElements
    (class Exercise)
    (relation isFor ?c)
    (property hasLearningContext ?el)
    (property hasCompetencyLevel ?ex_cl)
    (property hasDifficulty ?difficulty)
    (property hasCompetency ?competency)
    ))
  (assignIterator ?exercise ?exercises)
  (insertWithVariantsIfReady! ?exercise ?c)
  (insertElementOnce! ?exercise)
)

(:- (removeNotReady ?result ?concept ?exercises)
  ;; leave only those elements from ?exercises whose "for"-values
  ;; 1. are in the current course, OR
  ;; 2. are well-known, OR
  ;; 3. equal to ?concept
  
  ;;(call PrintDebug RemoveNotReady exercises ?exercises concept ?concept)
  (removeNotReady? ?result nil ?exercises ?concept)

  ;; base case: if there are no more exercises to process, then
  ;; return the result.
  (:first (}
(:first ((call PrintDebug RemoveNotReadyH Base case tempResult ?tempResult exercises ?exercises concept ?concept)
(same ?exercises nil)
(same ?result ?tempResult)))

(:first ((call PrintDebug RemoveNotReadyH case1 tempResult ?tempResult exercises ?exercises concept ?concept)
; eliminate the base case
(not (same ?exercises nil))
; Get the current exercise ?e.
(first ?e ?exercises)
(rest ?rest ?exercises)
; Get the concepts ?c for the current exercise ?e is for.
(assign ?cfor
(call GetElements ((class Fundamental) (relation inverseIsFor ?e))))
; Remove all concepts from ?c for that fit the above criteria.
(removeKnownConcepts ?cResult ?cfor ?concept)
; if the result list is empty, only then keep the exercise.
(same ?cResult nil)
(removeNotReadyH ?result (?e . ?tempResult) ?rest ?concept))
)

(:first ((call PrintDebug RemoveNotReadyH Base case2 tempResult ?tempResult exercises ?exercises concept ?concept)
(not (same ?exercises nil))
(first ?e ?exercises)
(rest ?rest ?exercises)
(assign ?cfor
(call GetElements ((class Fundamental) (relation inverseIsFor ?e))))
(removeKnownConcepts ?cResult ?cfor ?concept)
(not (same ?cResult nil))
(removeNotReadyH ?result ?tempResult ?rest ?concept))
)

(:- (removeKnownConcepts ?cResult ?cfor ?concept)
;; this lemma inverts the given list!
;; ?cResult stores the result, ?cfor are the concepts to test,
;; all concepts equal to ?concept are kept.
)

(:first ((call PrintDebug removeKnownConceptsH cfor ?cfor concept ?concept)
(first ?first ?cfor)
(rest ?rest ?cfor)
(removeKnownConceptsH ?cResult nil ?first ?rest ?concept))
)

(:first ((call PrintDebug removeKnownConceptsH base case tempResult ?tempResult c ?rest ?rest concept ?concept)
(same ?c nil)
(same ?result ?tempResult)))

(:first ((call PrintDebug removeKnownConceptsH case1 tempResult ?tempResult c ?rest ?rest concept ?concept)
(same ?c nil))
; test
(or (same ?c ?concept)
  (inserted ?c)
  (and (learnerProperty hasCompetencyLevel ?c ?cl)
    (call >= ?cl 3)))
)

;; if it fits, then do not insert it
(:call PrintDebug removeKnownConceptsH concept ?c wasn't inserted)
(first ?next ?rest)
(rest ?newRest ?rest)
)

(:first ((call PrintDebug removeKnownConceptsH case2 tempResult ?tempResult c ?rest ?rest concept ?concept)
(not (same ?c nil))
; We repeat the test to make sure it is never entered in case
; any earlier case was applied.
(not (or (same ?c ?concept) ...) \n\n©LeActiveMath Consortium 2006
(inserted ?c)
(and (learnerProperty hasCompetencyLevel ?c ?cl)
(call >= ?cl 3))
)

(defun removeKnownConceptsH (concept ?c inserted)
(first ?next ?rest)
(rest ?newRest ?rest)
)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defun queryUserInformation
(learnerProperty ?property ?var)
(学员 ?userId)
(queryUser ?userId for property ?property)
(same ?var (call LearnerProperty ?userId ?property))
)
)

(defun queryUserInformation
(learnerProperty ?property ?concept ?var)
(学员 ?userId)
(queryUser ?userId for property ?property on ?c)
(same ?var (call LearnerProperty ?userId ?property ?c))
)
)

(defun translateLearnerModelDataAndKnowledgeRepresentation
(equivalent ?cl elementary)
(call <= ?cl 1))
(equivalent ?cl simple_conceptual)
(call > ?cl 1)
(call <= ?cl 2))
(equivalent ?cl multi_step)
(call > ?cl 2)
(call <= ?cl 3))
(equivalent ?cl complex)
(call > ?cl 3))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(auxiliary axioms

; the following two axioms generate all possible binding of

; ?element to all the elements in the list provided as the as the
; second parameter. This is needed for "call GetElements".

; provide a binding to the first element of the list
(defun assignIterator ?element (?head . ?tail))
(same ?element ?head)
)
)

; recursion step
(defun assignIterator ?element (?head ?tail))
(same ?element ?tail)
.; A simple, but useful axiom.
(:- (same ?x ?x) (nil) )
(:- (first ?head ( ?head . ?rest)) (nil) )
(:- (first nil nil) (nil) )
(:- (rest ?rest ( ?head . ?rest)) (nil) )
(:- (rest nil nil) (nil) )
(:- (restrict ?list1 ?list2 ?result)
(same ?result
(call Restrict ?list1 ?list2)) )
(:- (removeElement ?result ?element ?list)
(removeH ?result nil ?element ?list))
(:- (remove ?result ?tempResult ?element nil)
(.:first

(same ?result
(call Reverse ?tempResult)))
)
)
(.:first

( removeH ?result ?tempResult ?first ?rest)
))
(:- (remove ?result ?tempResult ?element ( ?first . ?rest))
(.:first

(not (same ?first ?element))
))
)
(:- (getNonInserted ?result ?elements)

; (call PrintDebug getNonInserted ?elements)
(getNIH ?result (call Reverse ?elements) nil) )
(:- (getNIH ?result ?elements ?temp)
(.; base case: if nil, then return
(same ?elements nil)
(same ?result ?temp)
)
(.; case 1: the element is inserted
(first ?el ?elements)
(inserted ?el)
(rest ?rest ?elements)
(getNIH ?result ?rest ?temp))
(.; case 2: the element was not yet inserted
(first ?el ?elements)
(rest ?rest ?elements)
(getNIH ?result ?rest (?el . ?temp)) )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; learning tool support
(:- (learningServiceAvailable ?serviceName)
(call LearningServiceAvailable ?serviceName))

)