Deliverable N°: D31

Integration of Student Model in LEACTIVEMATH

Version 1

The LEACTIVEMATH Consortium

December 2005

Main Authors:

Paul Brna, Nicolas Van Labeke,
Rafael Morales, Helen Pain,
Kaśka Porayska-Pomsta

Project funded by the European Community under the Sixth Framework Programme for Research and Technological Development

Project ref. no.	IST-507826
Project title	LEACTIVEMATH- Language-Enhanced, User Adaptive, Interactive eLearning for Mathematics

Deliverable status	Restricted
Contractual date of delivery	December 31st 2005 (Month 24)
Actual date of delivery	January 31th 2006
Deliverable title	Integration of Student Model in LEACTIVEMATH
Type	Report
Status & version	1
Number of pages	24
WP contributing to the deliverable	WP4
WP/Task responsible	WP4 / 4.5
Author(s)	Paul Brna, Nicolas Van Labeke,Rafael Morales, Helen Pain, Kaśka Porayska-Pomsta
EC Project Officer	Colin Stewart
Keywords	Situational modelling, motivation, Bayesian networks.

Contents

Executive summary

This report provides a description of the integration of the Extended Learner Model (xLM) into LEACTIVEMATH, replacing the earlier ACTIVEMATH learner model component. It includes a brief description of xLM architecture in relation to LEACTIVEMATH and how information is exchanged between them via event messaging and procedure calls, followed by details of how each one of xLM components interact with LEACTIVEMATH.

xLM simple interface allows other LEACTIVEMATH components to request information about beliefs held in learner and situational models, including the evidence supporting them. They can also request decisions on what is the more likely status of learner states or dispositions, suggestions on how much autonomy and approval to give to learners and details of the learner history. For more content-oriented components, xLM is able to provide beliefs and decisions about learner capabilities and dispositions in relation to individual content items, such as an explanation or an exercise.

Most of xLM is integrated into LEACTIVEMATH even at the level of source code, meaning that most of xLM source code is included in LEACTIVEMATH source and it is compiled and deployed using the same mechanisms. The exception to the rule is the Situational Model component, whose source code is currently distributed, compiled and deployed independently.

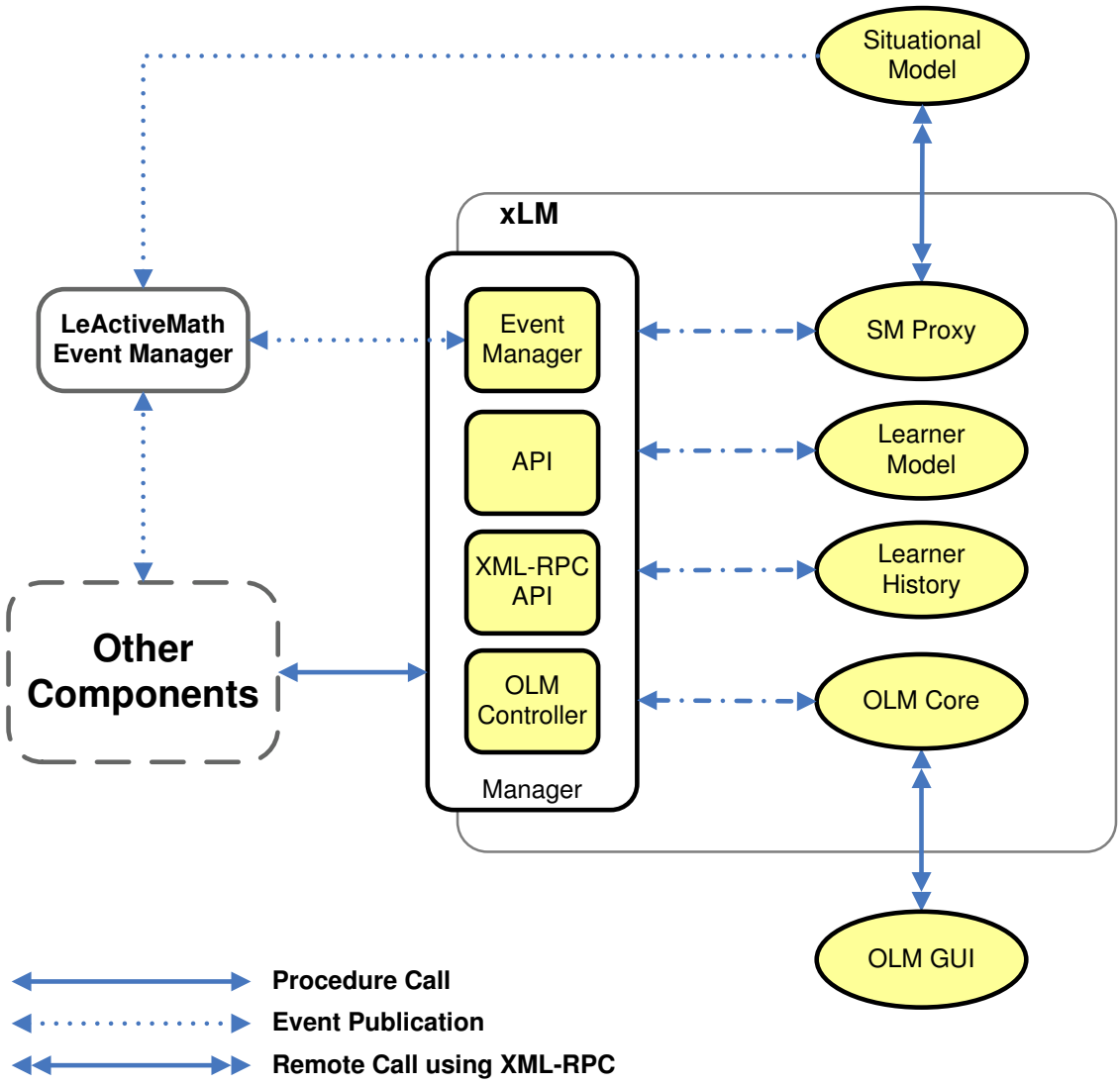

Figure 1: Overall architecture of xLM. Solid arrows represent procedure (method) calls and dotted arrows represent event publication. Double-headed arrows stand for remote procedure calls using XML-RPC.

1 xLM architecture and information exchange

The Extended Learner Model is composed of three big parts: diagnostic, open learner modelling and management. The diagnostic components and the management components of xLM are described in deliverable D30 (Andrès et al., 2005b). The Open Learner Model and related components are described in deliverable D29 (Brna et al., 2005). This report focuses on describing how xLM as a whole and each one of its core components—Situational Model, Learner Model, Open Learner Model and Learner History—integrate into LEACTIVEMATH.

The overall architecture of xLM is depicted in figure 1. The arrows connect-

ing components represent information interchange between components. There are two basic types of information interchange, as explained in deliverable D8 (Libbrecht et al., 2005): *procedure (method) calls*, represented as solid arrows, and *event publications*, represented by dotted arrows. In some cases information interchanges of both types occur between two components, and a dot-dash type of line is used to represent them in the figure.

- The main information interchange point between xLM and the rest of LEACTIVEMATH is the xLM Manager, which interchanges *events* (dotted line) with the LEACTIVEMATH Event Manager and provides a joint application programming interface (API) for the core diagnostic components of xLM: the Situational Model (SM), the Learner Model (LM) and the Learner History (LH). A subset of this API is available to remote LEACTIVEMATH components via XML-RPC, although xLM relies completely on LEACTIVEMATH for provision of connectivity.

- The xLM Manager acts as a local event manager for xLM. In other words, it collects all external events and distributes them locally, and almost all xLM components send their events to xLM Manager for it to distribute them both locally and to the LEACTIVEMATH Event Manager. The only exception is the Situational Model, which sends its events directly to the LEACTIVEMATH Event Manager (see below).

- Procedure calls between xLM components and other components in LEACTIVEMATH are indicated in figure 1 by the solid arrow on the left of the xLM Manager. Nevertheless, access to LEACTIVEMATH components is more spread inside xLM than showed in the figure. Details of it will be provided in sections devoted to the individual components of xLM, when relevant for explanations of their integration into LEACTIVEMATH.

- The Situational Model (SM) is executed as an independent process, not necessarily on the same computer where xLM is executed. SM integration into xLM happens through a proxy inside xLM that receives all information and requests send to SM. On the other hand, SM sends its events directly to the LEACTIVEMATH Event Manager.

- The Open Learner Model graphical user interface (OLM GUI) is delivered to be executed on the learner's computer, which is generally distinct from the server that executes LEACTIVEMATH, xLM and SM. However, a big part of OLM still runs on the server side—see (Brna et al., 2005) for details.

- All event interchange in between xLM components occurs through the xLM Manager, as depicted in figure 1. Information interchange via method calls also exists between xLM components, sometimes directly and other times indirectly through the xLM Manager API.

Table 1 summarises all the events currently implemented in LEACTIVEMATH and indicates which components of the Extended Learner Model fire or make use of them.

Table 1: LeActiveMath Events

Event	**LH**	**LM**	**SM**	**OLM**
Application Events				
ApplicationShutdownEvent	✔			
ApplicationStartupEvent	✔			
UrlRequestedEvent	✔			
UserCreatedEvent	✔	✔		
UserLoggedInEvent	✔	✔		
UserLoggedOutEvent	✔		✔	
UserPropertyChangedEvent	✔			
UserRemovedEvent	✔			
Interaction Events				
FocusChangedEvent	✔			
ItemPresentedEvent	✔			
ItemSeenEvent	✔			
WindowClosedEvent	✔			
Interaction (Dictionary) Events				
DictSearchedEvent	✔			
Interaction (Exercise) Events				
ExerciseActionEvent	✔			
ExerciseFinishedEvent	✔	✔	✔	
ExerciseHelpRequestEvent	✔			
ExerciseHintProvisionEvent	✔			
ExerciseStartedEvent	✔		✔	
ExerciseStepEvent	✔		✔	
Interaction (Feedback) Events				
HappinessEvent	✔			
SelfAssessmentEvent	✔			
SelfReportEvent	✔	✔		
Interaction (Book) Events				
PagePresentedEvent	✔			

(continued on next page)

✔Event intercepted (and processed) by the component
⊸Event fired by the component

Table 1: LeActiveMath Events (continued)

(continued from previous page)

Event	**LH**	**LM**	**SM**	**OLM**
UserBookDeletedEvent	✔			
UserBookPlannedEvent	✔		✔	
UserBookRenamedEvent	✔			
Content Events				
MBaseCollectionsChangedEvent	✔			
Tutorial Component Events				
TutorialInteractionEvent	✔			
xLM Component Events				
BeliefUpdatedEvent	✔	⊷		
SituationFactorChangedEvent	✔	✔	⊷	
OLMChallengeEvent	✔	✔		⊷
OLMMetacogEvent	✔	✔		⊷
OLMMoveEvent	✔	✔		⊷

✔Event intercepted (and processed) by the component
⊷Event fired by the component

2 Learner Model

The Learner Model (LM) component of xLM is the one in charge of maintaining models of learners as they evolve through their interactions with LEACTIVEMATH. LM takes content metadata and event information as input to produce updated beliefs about learners as output. Consequently, the main interactions between LM and the rest of LEACTIVEMATH occurs precisely at the gathering of input information and delivering of beliefs as products.

2.1 Events

Learner models are maintained by LM on the basis of information concerning learner behaviour that comes into xLM as event messages—see section 1 and deliverable D8 (Libbrecht et al., 2005). As all xLM components, LM subscribes to the xLM manager for its provision of events, which currently consist of events of types UserCreated, UserLoggedIn, ExerciseFinished, SelfReport, SituationFactorChanged and XlmOLMMove. Events of the first four types are produced by LEACTIVEMATH components outside xLM, while events of the last two types are produced inside xLM.

- UserCreated and UserLoggedIn events are interpreted by LM as request for creating a learner model for the learner just registered or logged into the system.
- ExerciseFinished events are used by LM for updating beliefs on learners' mathematical competencies.
- SelfReport events are used by LM for updating beliefs on learner's affective dispositions towards subject domain topics and mathematical competencies.
- SituationFactorChanged events are source of information for LM to update beliefs on learner's motivational dispositions towards subject domain topics and mathematical competencies.
- XlmOLMMove events are notifications of the status of beliefs in the discussion between learners and the Open Learner Model.

2.2 Metadata

Events of types ExerciseFinished, SelfReport and SituationFactorChanged are produced as a result of learners interacting with LEACTIVEMATH content items, and they all include an item identifier, which is used by LM to query LEACTIVEMATH Content Manager for the item's metadata. The metadata currently in use by LM includes

- associations between the item and other content items,
- mathematical competencies trained or tested by the item,
- competency level the item was designed for, and
- estimated difficulty of the item.

2.3 Beliefs

Beliefs are made available to the rest of xLM through a simple application programming interface (API) which the xLM Manager makes available to the rest of LEACTIVEMATH (section 1).

LM provides basic facilities for managing learner models, as instances of the class LearnerModel.

```
public void createLearnerModel( String learnerId);

public boolean existLearnerModel( String learnerId);

public void destroyLearnerModel( String learnerId);

public LearnerModel getLearnerModel( String learnerId);
```

In addition, LM provides facilities for accessing individual elements in the models, or *beliefs*, which are implemented as instances of the class Belief. To request a belief, a LEACTIVEMATH component needs to provide a learner identifier and a *belief descriptor*. The latter specifies the belief "coordinates" in the learner modelling space. The entries in the belief descriptor should be selected from the maps specifying the learner modelling dimensions, as specified in deliverable D30 (Andrès et al., 2005b), otherwise LM responds with a belief standing for complete ignorance.

```
public Belief getBelief( String learnerId, BeliefDescriptor descriptor);
```

Due to the fact that LEACTIVEMATH components are most frequently interested in "summaries" of beliefs, which are in essence decisions on what is most likely the learner case, and LM provides them as *summary beliefs*, which still include some measure of uncertainty, or *learner levels*, which are hard bets with no trace of uncertainty.

```
public double getSummaryBelief( String learnerId,
        BeliefDescriptor descriptor);

public int getSummaryLevel( String learnerId,
        BeliefDescriptor descriptor);
```

Beliefs in xLM learner models include the evidence supporting them. Although it can be requested directly from a belief which has been recovered via getBelief, LM supplies it on request via the following method.

```
public EvidenceSet getEvidence( String learnerId,
        BeliefDescriptor descriptor );
```

Some LEACTIVEMATH components need to know of learners qualifications and dispositions in relation to content items—e.g. *the motivation a learner may have towards doing exercise X.* This service is provided also by LM, which produces a belief (or summary of it) by taking into account the characteristics of the content item as described in its metadata.

```
public Belief getBelief( String learnerId, String itemId);

public Belief getSummaryBelief( String learnerId, String itemId);

public Belief getSummaryLevel( String learnerId, String itemId);

public Belief getBelief( String learnerId,
        String dimensionId, String itemId);

public Belief getSummaryBelief( String learnerId,
        String dimensionId, String itemId);

public Belief getSummaryLevel( String learnerId,
        String dimensionId, String itemId);
```

The first three methods above cover the most common case of a LEACTIVEMATH component requesting a belief, summary or decision on mathematical competency with respect to a piece of content—these methods deliver the closest to the *mastery* value produced by the earlier ACTIVEMATH Student Model component (see section 6.1. The last three methods generalise the previous case to recovering beliefs, summaries or decisions on any single dimension of the learner model (i.e. specific competencies, motivational and affective dispositions and metacognitive skills).

In addition to the core diagnostic functionality described above, LM includes facilities for recovering the individual maps that define the internal structure of its learner models. The LM API includes methods of the form

```
public DimensionMap getMapDimension ();
```

where *Dimension* is a shorthand for any of the learner modelling dimensions (section 2): Metacog, Affect, Motivation, Competency, CAPEs and Domain.

3 Open Learner Model

As described in Deliverable D29, the Open Learner Model is composed of three distinct parts:

- the OLM-GUI, located on the client-side of LEACTIVEMATH and responsible for presenting the collected information to learners and managing their exploration of their model. It is basically an applet, embedded in a browser window and deployed when requested by the appropriate URL.
- the OLM-Core, located on the server-side of LEACTIVEMATH and responsible for collecting information from the Learner Model and communicating with LEACTIVEMATH. It basically acts as an XML-RPC server with which the OLM-GUI communicates as a client. This communication is channelled through a couple of "handlers" implementing the various tasks the OLM-Core provides the OLM-GUI with.
- the OLM-Controller, also located on the server-side of LEACTIVEMATH and responsible for coordinating the actions of the learner in the OLM-GUI with responses in the OLM-Core. It is a MAVERICK [1] object whose task is to interpret and handle the relevant HTTP requests, build the appropriate model and pass on the URL parameters—if any.

3.1 Communications with LEACTIVEMATH

The communication inside the OLM (i.e. between the OLM-Core and the OLM-GUI) is ensured by the XML-RPC protocol. Communication between the OLM and LEACTIVEMATH is ensured by the event framework supplied by LEACTIVEMATH (see Deliverable D8, Libbrecht et al., 2005) and remains mostly located within the xLM.

The front-end of the xLM is the XLMEventManager, whose job is to receive the event published by LEACTIVEMATH and to dispatch them into the various sub-components of the xLM. It also act, in a similar way, for the events generated by the sub-components. This is the mechanism used for the OLM to communicate with the xLM.

Three different events are generated by the OLM, to be intercepted and interpreted by the Learner Model:

- OLMMetacog events are published when evidence of the metacognitive abilities of the learner are detected through their interaction with the OLM. This diagnosis is made on the basis of the dialogue moves performed by the learner, taking into account their nature (e.g. for exploration purpose like SHOWME or for challenge purpose like DISAGREE), repetition and context to refine the evidence.

[1] MAVERICK, see `http://mav.sourceforge.net/`

- OLMChallenge events are published by the OLM every time a "challenge" on its judgement is made by the learner. Here, "challenge" has to be taken in a broad sense, including all situations where the learner agrees with the judgement, where the learner disagrees with it and where the learner decides to "move on" and refuses to commit himself. An OLMChallenge event provides the Learner Model with a new piece of evidence about the learner's ability on the challenged topic. The nature of the evidence—and how it will be taken into account for updating the belief—depends on how (agreement, disagreement, avoidance) and where (on the claim or on one of the warrant/backing justifying it) the challenge was performed by the learner.
- OLMMove events are published by the OLM every time a dialogue move is made, either by the learner or by the OLM.

Tables 2, 3 and 4 describe the attributes defining both events.

Table 2: Attributes of OLMMetacog events

String moveID	The identifier of the dialogue move
Vector descriptor	The belief descriptor, target for the inference of the learner's metacognition
double depth	An estimation of the depth of the monitoring/control on this belief (should be normalised, so with a value between 0 and 1)
int initiative	Indicates if the move has been made on the learner's own initiative (1) or on the OLM's suggestion (0)

3.2 Integration with LEACTIVEMATH Front-end

As mentioned above, the bulk of the communication with LEACTIVEMATH will take place within the boundaries of the xLM—essentially, it is communication with the Learner Model—using the event framework. Nevertheless, there are two situations where the OLM and LEACTIVEMATH are cooperating directly: the deployment of the OLM from the front-end and the suggestion made by the OLM to the Tutorial Component for proposing further content to present to the learner.

3.2.1 Deployment of the OLM

The MAVERICK-VELOCITY approach used for structuring and implementing the Open Learner Model makes it very easy to implement its deployment at the front-end level, even ensuring a mixed-initiative strategy, where both the learners and LEACTIVEMATH can request this deployment.

Table 3: Attributes of OLMChallenge events

String moveID	The identifier of the dialogue move
Vector descriptor	The belief descriptor, target of the learner's challenge
String target	The target of the challenge, i.e. one of the Toulmin's element (claim, backing, etc.)
double confidence	Contains the confidence of the learner's challenge (between 0.1 and 0.9, since we assume the learner to be neither totally confident nor totally uncertain)
double intransigence	Contains the degree of intransigence of the learner, ie how much he/she is prepared to compromise with the OLM
double level	If the target is CLAIM, contains the level that the learner thinks represents his/her true abilities
int evidence	If the target is WARRANT or BACKING, it contains the index of the evidence (in the belief) that has been challenged. Otherwise, it contains -1
String attribute	If the target is BACKING, it contains the attribute of the event that has been challenged (eg "difficulty" of an exercise, "pride" from the SRT, etc)
String value	If the target is BACKING, it contains the value of the attribute that the learner thinks should be taken into account by the LM (eg "very_easy" for "difficulty").

Table 4: Attributes of the OLMMove events

String moveID	The identifier of the dialogue move
boolean isOLM	Indicates if the move has been played by the OLM (true) or by the learner (false).
Vector descriptor	The belief descriptor, target of the dialogue move.
String target	The target of the move, i.e. one of the Toulmin's element (claim, backing, etc.) or null if not applicable.

The Open Learner Model is uniformly accessed by its URL, as specified above. It is only on activation that instances of the OLM-GUI and OLM-Core will be created for each learner individually (and in a transparent way). This means that the OLM can be deployed both manually by the learner (see for example figure 2 showing the main menu of LEACTIVEMATH with a direct shortcut for the OLM) and programmatically by the system, using a single explicit URL.

The Tutorial Component for example is already using such possibility when implementing the tutorial strategies (see Deliverable D20, Reiss et al., 2005). One of the tasks of the strategies can be introduced to explicitly request learners to

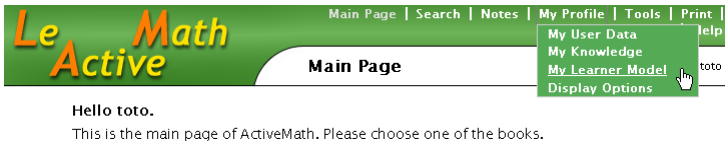

Figure 2: Snapshot of LEACTIVEMATH main menu, containing a shortcut to the OLM

access their model (for example at the end of the sequence). This task, besides introductory texts and prompts, contains a shortcut to the OLM. Depending on the circumstance, parameters can be added to the plain URL to specify the topic with which the OLM should be first deployed (see figure 3).

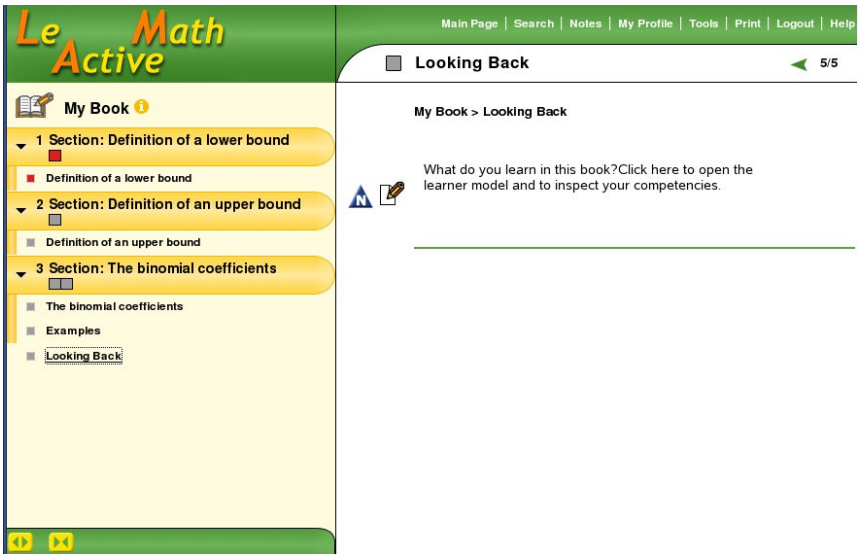

Figure 3: Snapshot of a learning object containing a reflection task, used to deploy the OLM

3.2.2 Suggestions to the Tutorial Component

When the negotiation between the OLM and the learner on a topic reaches a point where no agreements could be made, the OLM has the possibility to suggest to the learner to perform more exercises before resuming the discussion (the assumption being that more evidence gathered by the Learner Model may lead either the OLM or the learner to change their position on the disagreement).

Such a possibility is offered by the Tutorial Component—with its mechanism for selecting pieces of content (see Deliverable D20, Reiss et al., 2005))—and is providing us with an interesting bridge between the tutorial and self-reflective aspect of LEACTIVEMATH.

This functionality has been formally agreed between WP4 (OLM) and WP3 (Tutorial Component) and is in its early stage of implementation (most of the re-

maining investigations are focusing on which criteria are using to select the extra exercises, how they are presented to the learner and how they are referred to by the Learner Model).

4 Situational Model

As it is described in D30 (Andrès et al., 2005b), the Situational Model consists of three components:

- the Situational Diagnosis Agent (SDA),
- the Situational Modeller (SM-ler), and
- the Situational Model Proxy (SM-proxy).

The SDA is responsible for diagnosing the values of the situational variables such as learner confidence, interest, effort, and aptitude based on the information received as part of LEACTIVEMATH events. The SM-ler is responsible for conducting the diagnisis of the autonomy and approval values based in the situational diagnosis performed by the SDA. Finally, the SM-proxy is in charge of connecting the remote components of the Situational Model (SDA and SM-ler) with the rest of xLM.

As is the case with all sub-components of the xLM, the communication between the Situational Model and LEACTIVEMATH is facilitated by the event framework supported by LEACTIVEMATH (see Deliverable D8 Libbrecht et al., 2005). All three components of SM are in their own way involved in communication with LEACTIVEMATH.

The XlmEventManager—the front-end of xLM—receives LEACTIVEMATH published events and dispatches them to the Situational Model amongst other subcomponents of the xLM. The XlmEventManager also provides a gateway for the events generated by the xLM subcomponents.

The mechanism for communicating with the xLM and the rest of LEACTIVEMATH is supported by the Situational Model by means of event handlers. There are five LEACTIVEMATH types of events that the situational model handles:

- ExerciseStarted events inform the SM that a new exercise was started. They are used to provide information about the *difficulty* of the materials. SM also creates a copy of a Bayesian network for every learner for which such a copy does not already exist.
- ExerciseFinished events, which are used to infer the composites of the values of learner achievement, learner aptitude, interest and effort. They are some of the events that trigger the diagnosis of the situational variables to be sent to the xLM. They carry the information about the *success rate* of the solutions provided by the learners.
- ExerciseStep events, which provide information as to the *difficulty of exercises* and the learners' *success rates*. Just like ExerciseFinished events, they are events that trigger the diagnosed values of the situational variables to be sent to the xLM.

- UserBookPlanned events, which provide information as to the importance of materials. They are used only by the Importance object which sets the importance value to *high* if the Scenario type input value from the events is ExamSimulation.
- UserLoggedOut events, which are used to inform the Situational Model about the fact that the system is no longer in use by a specific user. These events are used to ensure that system maintenance is carried out and that copies of message buffers and copies of networks created for individual users at run-time are deleted once the users log out.

The event handling classes all implement XmlRpcServerMethod which allows their instances to be registered with the SM XmlRpcServer object created at the start of SM execution. This server object binds itself to a port (currently 27777) and listens for incoming traffic from the SM-proxy. In turn, the SM-proxy subscribes to the xLM Event Manager for event publication and re-sends the events it receives to the XmlRpcServer singleton in SM. When an event is in the end received by SM, its server object works out which of the XmlRpcServerMethod objects it contains should be invoked. The chosen event handler translates the incoming messages from XmlRpcValues to AMEvents, which then get passed into SDA for processing. The Situational Model internal processing involves calling DiagnoseFactorValues and GetSituationalDiagnosis methods on SDA.

The factor values calculated by the SDA are passed back to LEACTIVEMATH by instantiating an XmlRpcClient object, wrapping the factor values in an XmlRpcValue structure and passing this structure to the execute method on the XmlRpcClient. When LEACTIVEMATH Event Manager receives the XML-RPC call, it takes the information delivered by SM, fills a SituationFactorChanged event object with it and publish the event to its subscribers.

The situational diagnosis produced by SDA is also passed on to the NetworkManager which is responsible for running the SM-ler Bayesian network in order to calculate recommended values for autonomy and approval. A special type of events, GetFace, is used by the SM proxy to recover information from the SM-ler. Events of this type are empty, faked events send to the SM XmlRpcServer object every time there is request for the current recommendations for autonomy and approval. The results of running the Bayesian network are passed back to SM-proxy via the return parameter of the event handler for GetFace events.

5 Learner History

The Learner History needs to store all events received by xLM that are of relevance for learner modelling. However, the current implementation has gone beyond that, and currently stores all events produced inside LEACTIVEMATH, internal and external to xLM.

The learner history implementation is based on HIBERNATE, a framework for object/relational persistence with a query language (HQL) that is independent of the database in use. Furthermore, an important feature of HIBERNATE is the retrieval of stored objects by using a *criteria*-based approach, by setting constraints on attribute values that the objects retrieved must match.

To map objects to a database schema, an XML-based configuration is required that specifies which object to map to which table, and which field has to be mapped to which column. HIBERNATE will then take care of creating the schema if it does not yet exist in the database—more details of this can be found in deliverables D10 (Andrès et al., 2005a) and D30 (Andrès et al., 2005b).

Information can be recovered from LH either by specifying an identifier for an event or by constructing a query—a HistoryQuery. In the latter case, filters must be specified for the properties that events must have (including filters) and properties they must not have (excluding filters). Limits on event indexes in the list of events—ordered chronologically—and maximum number of events to be retrieved can be specified, as well as requesting only the number of events that match the query.

```
public ActivemathEvent fetchEvent(long id);

public List getHistoryEntries(HistoryQuery query);

public List getHistoryEntries(HistoryQuery query,
        int firstResult , int maxResults);

public int getNumResults(HistoryQuery query);
```

LH provides an additional method as a shorthand for the common query on whether a learner has already seen or not a content item.

```
boolean alreadySeen( String learnerId, String itemId );
```

6 The view from LEACTIVEMATH

This section summarises the integration of the Extended Learner Model (xLM) from the perspective of the rest of LEACTIVEMATH. While most of the integration work involves embedding the learner model into the existing LEACTIVEMATH application seamlessly, there are two notable places where the rest of the LEACTIVEMATH needs to be explicitly aware of the xLM: the *knowledge indicators* of the GUI, and the Tutorial Component, which uses the learner model for course planning.

6.1 Integration with the LEACTIVEMATH front-end and GUI

Since the xLM uses the OLM as its primary GUI, the rest of LEACTIVEMATH's user interface is mostly unaffected by the integration. The important exception here are the colored little boxes that LEACTIVEMATH uses to indicate the personal "knowledge" to its users (the term knowledge is used in a broad and general sense here).

The knowledge indicators are attached to each content item, book page, chapter and entire books. Depending on the knowledge level, they vary from grey (*unknown*) to red (*low*), yellow and green (*high*). Calculation of the knowledge level is based on single numerical value that the learner model provides for each user for a specific content item. Prior to xLM, the knowledge level was based on a "mastery value" computed by the existing learner model of ACTIVEMATH. With the integration of xLM, these values come from the LM and represent competency levels.

The basic function for integration is

double getSummaryBelief(String learnerId, String itemId)

which returns the desired single numerical value at the item level.

To fetch a learner's knowledge level for a content item, all LEACTIVEMATH components by design use a single method of the central User object representing the learner. It is only within this central method that the rest of LEACTIVEMATH front-end accesses the xLM to retrieve the knowledge level. By changing this method to access xLM instead of the old learner model, integration into the front-end was easy to achieve.

6.2 Integration by the Tutorial Component

The Tutorial Component, especially its central sub-component, the course generator, uses the information provided by the xLM extensively. Deliverable D20, Formalized Pedagogical Strategies (Reiss et al., 2005), specified the pedagogical knowledge underlying the course generation process, and in particular the relationship between the current state of the learner as represented in the xLM and the selection of the content items to be presented in a course.

The pedagogical strategies are implemented using JSHOP2, a hierarchical task network planner. The following lines of code provide an example of the implementation:

```
(:method (illustrateWithSingleExample! ?c)
         (;; IF
          (learnerProperty hasField ?field)
          (learnerProperty hasEducationalLevel ?el)
          (learnerProperty hasMotivation ?c ?m)
          (call < ?m 2)
          (learnerProperty hasCompetencyLevel ?c ?cl)
          (equivalent ?cl ?ex_cl)
          (assignIterator ?example
                          (call
                           GetElements
                           ((class Example)
                            (relation isFor ?c)
                            (property hasLearningContext ?el)
                            (property hasCompetencyLevel ?ex_cl)
                            (property hasField ?field)
                            )))
          )
          ;; THEN
         ((insertElementOnce! ?example))
         )
```

This method is one of several methods that try to fulfill the pedagogical tasks of illustrating a concept using a single example. The upper part of the method specifies the methods preconditions that have to be fulfilled in order for the method to be applicable. It basically states that if the learner's motivation is low (< 2) and there is a content element available that is an example of the field of the learner and its learning context corresponds to the educational level of the learner, and its competency level corresponds to the current competency level of the learner, then this example will be added in the course (the actions in the bottom part of the method). A similar method encodes that in case the learner is highly motivated, then one can insert an example requiring a higher competency level. The course generator accesses the information represented in the xLM via a bridge that hides the concrete xLM implementation from the planner. This bridge takes the queries from the course generator and builds the corresponding believe descriptors (see below for a code snippet that generates a belief descriptor accessing metacognitive information).

```
public double getMetacognition( String userId, String itemId,
        Metacognition meta)
{
    try {
```

```
            double result = doGetSummaryBelief(userId,
                    meta.toString(), itemId);
        }
        catch (\xLMException e) {
            log.error( "Couldn't retrieve belief for "
                + "metacognition dimension '" + meta + "' on "
                + itemId);
            return -1;
        }
        return result;
    }
```

7 Outstanding issues

From the beginning, the Extended Learner Model (xLM) has been developed as an integral part of LEACTIVEMATH. However, some care has been taken to make xLM easily detachable from LEACTIVEMATH in the future. Accordingly, some issues regarding the integration of xLM into LEACTIVEMATH have already been discussed in deliverable D30 (Andrès et al., 2005b), so only a summary is given below.

Limited coverage of content Due to restrictions on the use and adjustment of OMDoc, the language used in LEACTIVEMATH for representing mathematical content, a separate concept map for the subject domain has been the implementation of choice for xLM. It provides a solid ground for learner modelling which is less sensitive to repetitions, inconsistencies, errors and changes in content. Despite the WP4 team's attempts to make the relevance of this map clear to the project, and the value of this more ontologically oriented approach to the subject domain, any efforts in this direction have been exerted with minimal coordination between work packages. The map of the subject domain has been seen as exclusive to the interests of xLM, and as a consequence has had almost no support elsewhere in the project. The subject domain map of Differential Calculus, as currently implemented as part of xLM, is certainly adequate for LEACTIVEMATH evaluation but some further work is needed by the project.

Lack of support for misconceptions Misconceptions identified and annotated by authors have been formally introduced in content and content metadata only very recently, but their representation is still very primitive – a list of OMDoc symbols with no references to other content items. This condition has severely affected xLM processing of misconceptions. The framework is set for it, yet no further details of their processing could be implemented.

Competency and mastery Both content and xLM has been developed on the basis of a new view for mathematical teaching and assessment grounded on the notions of mathematical competencies and competency levels (OECD, 2003; BMBF, 2004). However, changes to LEACTIVEMATH front-end to upgrade it to the new language is still limited to replacing the earlier notion of *mastery* by the new notion of *overall mathematical competency*. Although access to the full new scheme is available to users through the OLM GUI, the production of explanations that are readily accessible to learners is still ongoing work.

References

Andrès, E., P. Brna, N. van Labeke, M. Mavrikis, R. Morales, H. Pain and K. Porayska-Pomsta (2005a). Student model specification. Deliverable D10, LeActiveMath Consortium.

Andrès, E., P. Brna, N. van Labeke, R. Morales, H. Pain and K. Porayska-Pomsta (2005b). Diagnostic functionalities. Deliverable D30, LeActiveMath Consortium.

BMBF (2004). The development of national educational standards: An expertise. Tech. Rep. 1, Federal Ministry of Education and Research, Berlin, Germany.

Brna, P., N. van Labeke and R. Morales (2005). Open student model. Deliverable D29, LeActiveMath Consortium.

Libbrecht, P., E. Andrès, O. Lemon, R. Morales, K. Porayska-Pomsta, S. Winterstein, C. Ullrich and C. Zinn (2005). Open architecture. Deliverable D8, LeActiveMath Consortium.

OECD (2003). *The PISA 2003 Assessment Framework*. Organisation for Economic Co-Operation and Development.

Reiss, K., M. Moormann and C. Groß (2005). Formalized pedagogical strategies. Deliverable D20, LeActiveMath Consortium.