Deliverable N°: D30

Diagnostic Functionalities

The LEACTIVEMATH Consortium

December2005

Version 1

Main Authors:

Eric Andrès, 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	Diagnostic Functionality
Type	Prototype
Status & version	1
Number of pages	66
WP contributing to the deliverable	WP4
WP/Task responsible	WP4 / 4.4
Author(s)	Eric Andrès, Paul Brna, Nicolas Van Labeke,Rafael Morales, Helen Pain, Kaśka Porayska-Pomsta
EC Project Officer	Colin Stewart
Keywords	Learner modelling, situational modelling, beliefs, belief function, Dempster-Shafer Theory.

Contents

Executive summary

This report is part of deliverable D30 *Report and Prototype of Diagnostic Functionalities* and provides a description of the implementation of the diagnostic functionalities of the Extended Learner Model (xLM), the prototype learner modelling subsystem of LEACTIVEMATH that replaces the earlier ACTIVEMATH learner model. It includes also an explanation of the rationale behind the design and implementation of xLM, and a discussion of outstanding issues.

The diagnostic functionalities of xLM are built on top of a concept map that explicitly tries to represent the semantic structure of the subject domain and LEACTIVEMATH content. The formal description of the underlying structure of a mathematical domain is complemented with the operationalisation of a new standard for supporting and evaluating mathematical learning, based on the notions of mathematical competency and competency level. In addition, xLM is able to diagnose and model a variety of emotional, attitudinal, motivational and situational aspects of learning. Furthermore, by integrating the Open Learner Model delivered in D29 *Report and Prototype of Open Student Model*, xLM is able also to diagnose some aspects of the meta-cognitive skills of learners.

The inferencing capabilities of xLM are developed on top of Bayesian inference and a variation of Dempster-Shafer Theory known as the Transferable Belief Model. The former has a short but solid tradition in learner modelling, and the latter offers the opportunity to model a broader sense of uncertainty about the learner and deal better with conflicting evidence.

xLM simple interface allows other LEACTIVEMATH components to request information about beliefs held in learner 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.

Chapter 1

Introduction

LEACTIVEMATH fits the description given by the Advanced Distributed Learning Initiative for a second-generation e-learning system that combines a modern content-based approach from Computer Based Instruction (CBT)—in its book metaphor, developed on top of web technology and learning objects—with adaptive educational strategies from Intelligent Tutoring Systems (ITS) (Polson and Richardson, 1988; ADL, 2004).

This mixture of approaches have produced tensions in the design of LEACTIVEMATH in general, but particularly in the design of the Extended Learner Model (xLM), LEACTIVEMATH learner modelling subsystem. xLM is aimed at supporting a wide range of adaptive educational strategies, from coarse-grain book construction to tailored natural language dialogue, without the support of painfully designed and dynamically constructed learning activities capable of providing large amounts of specific and detailed information about learner behaviour—something traditionally afforded by ITS systems. LEACTIVEMATH makes heavy use of pre-authored educational *content* to support learning, hoping to capitalise in this way on the expertise of a variety of authors at producing educational materials. However, educational content is for the most part opaque to learner modelling, in the absence of domain expert subsystems to query about what is inside it. The information available for learner modelling is hence reduced to what has been explicitly provided by authors in the form of *metadata*.

Metadata is a heavy burden on authors since it amounts to work be done twice: to say something and to say what it was said. The more detailed and accurate the metadata, the more extra work to do. Consequently, metadata tends to be subjective and shallow, with a well-intentioned drive towards standardisation thwarted (from a modelling point of view) by the shallowness of current metadata standards such as LOM (IEEE, 2002).

Guidance to learners through educational content in many e-learning systems, as in LEACTIVEMATH, jumps in between two extremes: predefined paths and content browsing. From a learner modelling perspective, both situations are for the most part equivalent, since neither of them accommodates the presentation of new content materials to the modelling needs. Whereas in some ITS systems learner modelling can lead the learner's progress through the subject domain (e.g. Corbett and Anderson, 1995), in our case it has to be *opportunistic*, taking advantage of whatever information is available. To round the picture,

the learner modelling component of LEACTIVEMATH was required to model the learner meta-cognitive state, as well as their motivational and affective states and dispositions (LeActiveMath, 2004).

A learner modelling subsystem in this conditions has to do "more with less," answering questions about learners on the basis of scarce information, hopefully without pursuing blind over-generalisation. Nonetheless, this is not an uncommon situation at the onset of adaptive content-based e-learning systems. The consequences of the requirements and working conditions described above on the design and implementation of xLM are described in this report. They can be summarised in six aspects: a modular design for the learner modelling subsystem, a generic multi-dimensional modelling framework, tolerance to vague and inconsistent information, capability for representing both conflict and ignorance about learners, squeezing of sparse information and open learner modelling.

The modular design of xLM attempts to draw in its implementation the map of the distinct tasks that are involved in learner modelling: keeping a record of learner history, the raw data for learner modelling; building a model that explains this history and attempts to elucidate the present and the near future; modelling ongoing states as well as trends and dispositions, and making all this available for learner inspection and negotiation. xLM design prepares it to meet the requirement of a range of e-learning systems similar to LEACTIVEMATH. New needs can be meet by localised changes in individual components that become mostly transparent to the rest of the xLM subsystem.

The current diagnostic functionalities of xLM are built in accordance with a modern approach to supporting and evaluating mathematical learning that is based on mathematical competencies (OECD, 2003). Mathematical competencies in xLM aggregate into an orthogonal dimension that *applies* to the subject domain, the collection of topics of discourse of LEACTIVEMATH content. A similar approach for a collection of meta-cognitive, motivational, affective and situational aspects of learning, as orthogonal dimensions that apply to each other in a predefined way, gives xLM flexibility and power to diagnose and model a wide range of learner states and dispositions. Propagation of evidence according to the structure of each individual dimension is an additional feature of xLM that takes it further away from ACTIVEMATH old learner model. Propagation of evidence allows xLM to profit more from explicit knowledge on the semantic structure of the subject domain and other learner modelling dimensions, and from the information provided by other LEACTIVEMATH components.

Tolerance of vague and inconsistent information has been built into xLM by employing techniques of approximate reasoning. These are Bayesian inference, a technique of growing popularity for user and learner modelling (Conati et al., 2002; Jameson, 1996; Zapata-Rivera and Greer, 2000), and the Transferable Belief Model (TBM, Smets and Kennes, 1994), a variation of Dempster-Shafer Theory (DST, Shafer, 1976). TBM-like techniques have not been used for learner modelling so far, and have been scarcely used for user modelling (Jameson, 1996).

However, they seem more suitable to face the problem of learner modelling for the broad and varied community of web users because of its capability to formally model a broad sense of uncertainty about learners and to deal with conflicting evidence.

xLM makes its diagnostic functionality available to the rest of LEACTIVEMATH through a single entity, the xLM Manager, and a well defined application program interface (API). xLM receives requests from other LEACTIVEMATH components regarding beliefs held in learner models, which are delivered packaged with the evidence supporting them. LEACTIVEMATH components that do not need such a level of detail, or are still unable cope with it, can request summative versions of beliefs as numerical values that imply decisions on what is the more likely status of learner states or dispositions. Suggestions on how much autonomy and approval would benefit learners more, and details of the learner history are also available under request. Even for content-oriented components, which prefer to talk about learners in relation to specific content items, xLM is able to provide summative beliefs and decisions about learner capabilities and dispositions given a content item. This variety of generic mechanisms for accessing xLM information and inferences aims to makes it easily accessible to the rest of LEACTIVEMATH.

Chapter 2

Principles

2.1 Introduction

The design and implementation of the diagnostic functionality of the Extended Learner Model (xLM) are based on a number of principles. Overall, xLM architecture was designed and implemented following a principle of *modularity*. This means that the diagnostic functionality of xLM has been distributed among three of its core components: Situational Model (SM), Learner Model (LM) and Learner History (LH). The Situational Model diagnoses the learner's motivational *state* in the ongoing *situation*. The Learner Model collects SM diagnoses—as well as diagnoses provided by other components of LEACTIVEMATH—and reports of learner behaviour in order to diagnose the learner's long term *qualifications* and *dispositions* towards the subject domain. Finally, the Learner History keeps a record of all evidence supporting the other components' diagnoses.

The principle of modularity has been applied also to how xLM integrates with the rest of LEACTIVEMATH: there is a single main interchange point, the xLM Manager, that provides access to all xLM functionality. Within the limits of practicality, effort has been put into making of xLM a generic learner modelling subsystem and reducing its dependencies on LEACTIVEMATH, so that it should be relatively easier to decouple them in the future for xLM to serve other systems.

The following sections explain the principles behind the design and implementation of the Situational Model (section 2.2), the Learner Model (section 2.3) and the Learner History (section 2.4). The other core component of xLM, the Open Learner Model (OLM) is described in deliverable D29 (Brna et al., 2005).

2.2 The Situational Model

The design and implementation of the Situational Model relies largely on the results of the analysis of the empirical data gathered in the context of LEACTIVEMATH by means of three studies that were described in detail in the deliverable D10 (Andrès et al., 2005). The methodology on which the analysis of the studies was based was determined by the nature of the early design specification of the situational model given also in the deliverable D10.

2.2.1 Data analysis methodology

The overall data analysis methodology used in determining the design and implementation of the situation model was driven by the specification of the system and it resulted in two levels at which the analysis was performed:

1. Quantitative analysis of the numerical data was used to determine the relevant situational variables to be modelled in the SDA and to identify the relationships between the situational variables in specific combinations.
2. Qualitative analysis of the verbal protocols was used to determine the principles by which the relationships between the situational variables would impact on the autonomy and approval values calculated by the SM-ler.

Three types of quantitative analysis were carried out:

- Descriptive analysis which was used to obtain a measure of relative importance to tutors' feedback of each situational variable in the data set.
- Pearson's linear correlations analysis which was used to establish the relationships between the individual factors.
- Principle Factor Analysis (PFA) was used to determine the groupings of factors and to reduce the dimensionality of the data.

For the descriptive analysis we used frequency measure to determine the overall relevance of every variable to tutors' feedback decisions. This was necessary to make the design and especially the implementation of the Situation Model both feasible within, and relevant to LEACTIVEMATH. Specifically, as we reported in the deliverable D10, we wanted to find out **what** factors the tutors consider when deciding on their feedback and **how often** these factors are used across all interactions. The purpose of this was to establish a minimum set of situational factors that would constitute both necessary and sufficient input to the situation model.

The need to perform the correlational analysis was motivated by the fact that the situational model is defined in terms of high interdependence between a number of relevant situational variables and their values. For example the degree of student's confidence may depend on the degree of difficulty of the exercise or exercise step. Identifying the exact relationships between the situational variables and the strength of those relationships is central to the design of the situation model and to its ability to perform the diagnosis of autonomy and approval values.

Finally the Principle Factor Analysis was used to aid the reduction of complexity of the emerging model. High interdependence between many variables is generally difficult to model computationally. It is typically beneficial to reduce the dimensionality of the modelled space in order to increase the perspicuity of the

design and efficiency of the implemented system. The Principle Factor Analysis was also used to aid decisions as to whether or not to retain the variables between which the relationship was weaker.

With respect to the qualitative analysis, it consisted of two stages: (1) the preparation stage and (2) the actual analysis stage. In the preparation stage the data was clustered based on the groupings of the situational variables obtained from the PFA. The clustered data was then summarised in terms of the individual value combinations of the situational variables in each cluster. The value combinations in each cluster were identified by the *interaction ID* which was composed of tutor ID, student ID, situation ID and a time stamp indicating the date and time at which a particular situational variable value combination occurred in a given interaction. The summaries were used to guide the preliminary analysis of the logged dialogues along with the verbal protocols in terms of the degree of autonomy (guidance) and approval intended by the tutor in each relevant situation. Ultimately the qualitative data analysis served as an empirical basis for defining the rules which would enable the calculation of autonomy and approval values by the Situation Model.

2.2.2 Results of the analysis

As was already reported in D10, in order to understand the relevance of each factor used by the tutors in our studies to tutor feedback, we allowed the tutors to select from a range of clues, including the value "irrelevant" whenever it was applicable during their interactions with the students. During data analysis, we used these values to perform frequency counts to reveal the relative importance of each factor to tutors' decisions across all interactions. Figure 2.1 illustrates our findings.

We identified 5 groups of factors each with a decreasing relevance to tutors' decisions. The most important factors of which the frequency count (FC) is above 200 out of the total of 309 recorded situations include **correctness of student's answer** with FC = 259, **student's confidence** (FC = 250), **student's aptitude** (FC = 222), and **student's interest** (FC = 214).

The second group with the overall frequency count above 150 includes **difficulty of material** (FC = 183) and **importance of material** (FC = 165).

The third group of factors used over a 100 times includes **student effort** (FC = 140) and **amount of session time left** (FC = 119).

Two factors were use more than 50 times: **student's knowledge** (FC = 78) and **amount of material left to cover in the current session** (FC = 76).

The group of the least important factors, used less than 50 times by different tutors across interactions, includes **student's emotional state** (FC = 46), **relative difficulty (FC = 41)**, **interface** (FC = 38) and **goal** (FC = 37).

The results of the frequency counts show that not all the factors are equally important to tutors' decisions across the board. What these results give us is a sys-

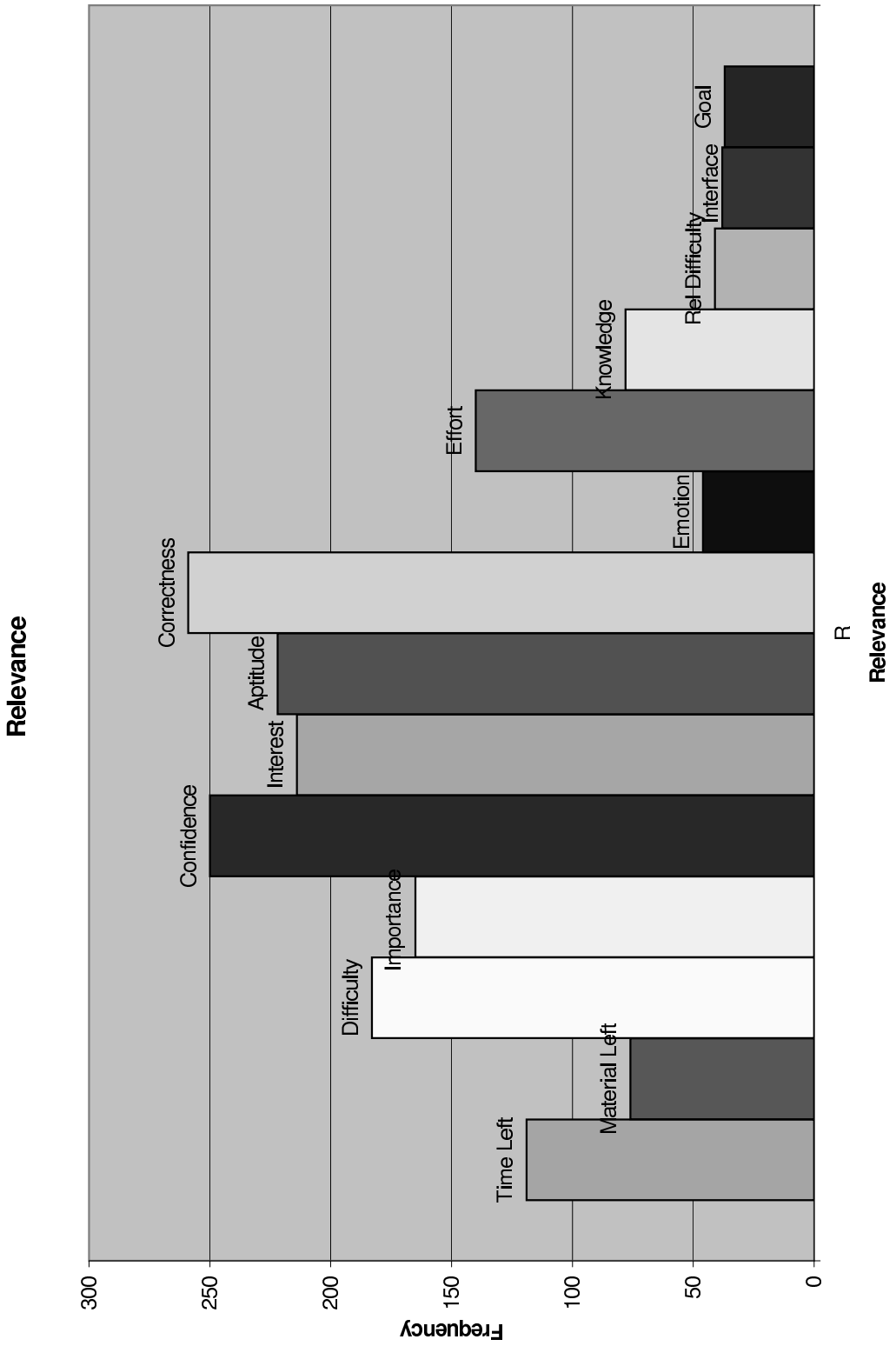


Figure 2.1: Relevance of individual factors to tutors' decisions

tematic, empirically supported way of selecting the factors which are essential to our model. These include most factors in the first, second, third and possibly the fourth group, but probably not in the fifth group where the factors were used between different tutors less than 50 times in total. Ultimately our decisions regarding the inclusion of a factor in our system rely primarily on whether or not further support for or against it can be found in other data sources that we gathered such as verbal protocols, interviews and cognitive walkthroughs. For example, the **goal** and **interface** were not only selected by the tutors very infrequently, but they were also used by only one out of five tutors who additionally found it virtually impossible to define the meaning of the two factors in a systematic way or to point to the sources of evidence in students' behaviour that could be used to infer such meaning. Without such information from the tutor it becomes very arbitrary an affair to define the way in which the values of the factors can be inferred, what values are relevant at all and how they may interact with other relevant factors. This is certainly a problem which we also encountered with **student's emotional state** factor which again was used very infrequently by the tutors.

The descriptive analysis allowed us to weed out all of the situational variables which were identified through the analysis as being used less than 50 times across all the interactions in our data set. This meant eliminating **student's emotional state**,**relative difficulty**, **interface** and **goal**. Furthermore, we examined the remaining set of situational variables with respect to relevance to the LEACTIVEMATH environment. This lead to the elimination of further two variables from the set, namely the **amount of time left** and the **amount of material left**. Whilst these two factors were used relatively frequently by the tutors in our study, they could not be used in the context of LEACTIVEMATH which neither imposes a time pressure on completion of exercises nor keeps track of the amount of material left till the end of a session. The resulting set consisted of eight situational variables which included:

1) student's confidence,
2) correctness of student's answer,
3) student's aptitude,
4) student's interest,
5) difficulty of an exercise,
6) importance of an exercise,
7) student's effort, and
8) student's knowledge

We used the data for the remaining set of situational variables to perform Pearson's linear correlations analysis to identify whether or not the variables affected each other in specific combinations and if so what was the nature of the relationship between them. We found that most situational variables were in a relationship with one another to various degrees. Only two sets of variables did not seem to be correlated. These were **student confidence** and **importance of mater-**

ial, and **correctness of student answer** and **the level of student knowledge**. The apparent lack of a relationship between the last pair is surprising as one would expect the level of student's knowledge to affect the correctness of their answers. Whilst it is difficult to find a strong explanation for this, it is possible that, simply, the student's knowledge factor was used was inconsistently by the tutors. This factor was mostly used by only one out of the 5 tutors that took part in our studies, and even that tutor had problems with providing a coherent definition of its meaning. Altogether and surprisingly, the student knowledge variable was used 78 times out of the total of 309 analysed presently.

As is shown in table 2.1, most relationships were substantial and were characterised by moderate correlations. This was indicated by both the significance levels and the values of the Pearson's *r* correlation coefficient. A less strong relationships, though still indicated by moderate correlations, were found between **correctness of student answer** and **student effort**, and between **student's effort** and **student knowledge**.

In order to gain a better view on the relationships between the factors, and to obtain a grouping of factors that would aid our implementation efforts, we also conducted a Principle Factor Analysis on the remaining data set. For this analysis we used Varimax rotation with Kaiser normalisation. The results consisted of 3 groups shown in table 2.2. Four situational variables were highly correlated with the first and the strongest group accounting for 34.7% of the variance in the data. In order of the strongest correlation with the group, the variables included **student effort** (.829), **student aptitude** (.826), **student interest** (.798) and **student confidence** (.660). Other factors clearly correlated with this group, but to a lesser extent. The strength of the remaining variables on other groups extracted through the analysis suggest that these factors should not be considered in group 1. These include **difficulty of material** and **importance of material** both strongly loading on group 2 with .771 and .901 loading values respectively.The second group accounted for 23.6% of variance. Finally, the third group (accounting for 17.7% of variance) had predominantly one significant factor loading – that of **correctness of student answer** with the loading value of .959.

Again a problematic factor is that of **student knowledge**. Although it loads on the first group with the value of .317, it is not significant enough to be included in this group: a typical Kaiser normalised cut off value for inclusion in a group is around .45. This means that the factor should not be included in either group 1 or group 3. On the other hand its loading on group 2 is negative (with the value -.636) indicating that in fact it is not relevant to that group at all.

Given that in our correlations analysis **student knowledge** has been found to be only weakly correlated with other factors, coupled with the lack of frequency with which tutors selected or commented on it during interactions, plus the point that actually none of the tutors were able to provide a clear definition of this variable, led us to a decision to remove this variable from the set to be modelled in LEACTIVEMATH. By removing this factor from the set analysed we reduced the

Table 2.1: 2-tailed Pearson's correlations coefficient matrix showing moderate correlations and substantial relationships.* Correlation is significant at the 0.05 level. ** Correlation is significant at the 0.01 level.

	Difficulty	**Importance of mat.**	**S' Confidence**	**S' Interest**	**S' Aptitude**	**Correctness**	**S' Effort**	**Knowledge**
Difficulty	1.000	.683**	.349**	.400**	.308**	.364**	.307**	-.446**
Importance of mat.		1.000	.105	.458**	.358**	.266**	.264**	.-.332**
S' Confidence			1.000	.538**	.579**	.244**	.325**	.127*
S' Interest				1.000	.585**	.210**	.523**	.222**
S' Aptitude					1.000	.283**	.555**	.194**
Correctness						1.000	.114*	.052
S' Effort							1.000	.128*
Knowledge								1.000

Table 2.2: Rotated Component Matrix. Extraction Method: Principle Component Analysis with Kaiser Normalisation. Rotation converged in 5 iterations.

	Rescaled Component		
	1	**2**	**3**
Difficulty	.305	.771	.261
Importance of mat.	.287	.901	.105
S′ Confidence	.660	-.072	.287
S′ Interest	.798	.227	.093
S′ Aptitude	.826	.069	.211
Correctness	.134	.137	.959
S′ Effort	.829	.086	-.112
Knowledge	.317	-.636	.101
% of Variance:	*34.7*	*23.6*	*17.7*

number of modelled variables from 8 to 7. We ran the Principle Factor Analysis again on the reduced data set. The result (shown in table 2.3) consisted of two clearly defined groups. The first group accounted for 35.6% of variance and included **student aptitude** (.829), **student confidence** (.777), **student interest** (.757), and **student effort** (.731). The second group accounted for 28.4% of variance and included **importance of material** (.864), **difficulty of material** (.850), and **correctness of student answer** (.576). The new groupings as well as the loadings of variables on the groups is statistically much more satisfactory as it provides clear indication of both the relationships between the variables and conforms to the tentative groupings of the situational factors proposed in the deliverable D10.

2.3 Learner Model

The Learner Model (LM) component of xLM was designed to model a wide range of learner qualifications, abilities and dispositions in relation to a subject domain, and to accommodate to a variety of web-based and content-based e-learning systems. Our approach to the first goal is based on general knowledge of the human

Table 2.3: Rotated Component Matrix. Extraction Method: Principle Component Analysis with Kaiser Normalisation. Rotation converged in 3 iterations.

	Rescaled Component	
	1	**2**
Difficulty	.230	.850
Importance of mat.	.154	.864
S′ Confidence	.777	.092
S′ Interest	.757	.339
S′ Aptitude	.829	.219
Correctness	.137	.576
S′ Effort	.731	.139
% of Variance:	*35.6*	*28.4*

cognitive and emotional architecture (Stillings et al., 1995; Ortony et al., 1988) and consists in structuring learner models as multi-dimensional, arranging their *dimensions* in a predefined way and representing qualifications and dispositions as the *application* of some dimensions upon others. This allows LM to hold beliefs on, for example, a learner's competency *on* on the subject domain, their motivational dispositions *towards* training or exercising their competencies *in* a subject domain, and their monitoring *of* their self-confidence *in relation to* a subject domain. The richness of the learner models depends in this way of the definition of the internal structure of each one of their dimensions and the availability of evidence to construct beliefs on their applications.

For the case of LEACTIVEMATH, the dimension corresponding to the subject domain is defined to cover a subset of Differential Calculus, qualifications are mathematical competencies (OECD, 2003), motivational dispositions were chosen on the basis of the results from the studies reported in (Andrès et al., 2005) and summarised in section 2.2, affective dispositions were suggested by mathematical educators in the project and meta-cognitive abilities have been been chosen from the literature (Brown, 1987).

The second goal, to make xLM general and flexible enough to meet the needs of a variety of web-based and content-base e-learning systems, is supported by the

multi-dimensional framework for learner models in the sense that the definitions of the internal structure of its dimensions can be tailored to the needs of each system, as it has been done for LEACTIVEMATH. In addition, representation of beliefs in learner models has been chosen to meet the fact that open web-based system can be reached by a wide variety of users, with such a diversity of backgrounds that it makes difficult to initialise their models to a better default other than complete ignorance on their characteristics. Belief representation in LM is hence based on the Transferable Belief Model (TBM) (Smets and Kennes, 1994), a modern incarnation of the classic Dempster-Shafer Theory (Shafer, 1976) which is able to represent uncertainty in beliefs, complete ignorance—no prior probabilities as in Bayesian models, more commonly used for learner modelling (Conati et al., 2002)—and amount of conflict in accumulated evidence.

2.4 Learner History

The learner history supports the diagnostic components by storing the learners' interactions with the system. The interactions are tagged with a unique identifier, thus allowing xLM to make justifiable its diagnostic reasoning. Furthermore, the learner history offers a sophisticated query functionality to enable the diagnostic components to make use of the learners' past actions in their reasoning process. For more details, please refer to deliverable D10 Student Model Specification (Andrès et al., 2005).

Chapter 3

Functionality

The Extended Learner Model (xLM) replaces in LEACTIVEMATH the learner model component of ACTIVEMATH. Accordingly, xLM main tasks are

- to diagnose learner states and dispositions in relation to the subject domain LEACTIVEMATH provides support for,
- to maintain a reliable model of them, as they evolve over time, and
- to deliver this information to other components in the system so that they can better support learning with LEACTIVEMATH.

The principles behind the design of xLM have been explained in chapter 2, including explanations of how xLM has been designed to diagnose and model learners. More details of these functionalities are presented in chapter 4. This chapter deals with the information on learner states and dispositions that is made available by xLM to other LEACTIVEMATH components.

3.1 Diagnosis

The Description of Work for LEACTIVEMATH specifies that xLM diagnostic functionality must include inferences of learners' domain knowledge, as well as motivational, emotional and situational aspects of their learning (LeActiveMath, 2004, Task 4.4 on p. 35). A selection of the results from xLM diagnosis of learners are made available to the rest of LEACTIVEMATH through xLM application program interface (API, section 3.3).

1. Learner's needs of *autonomy* and *approval* in the current *situation* of the interaction between the learner and a content item.
2. Learner's *competencies* on the topics in the subject domain.
3. Learner's *motivational* and *affective dispositions* towards the topics in the subject domain and towards competencies on the topics in the subject domain.
4. Learner's *meta-cognitive skills* in relation to
   (a) competencies on the topics in subject domain,
   (b) motivational and affective dispositions towards the topics in the subject domain, and

(c) motivational and affective dispositions towards competencies on the topics in the subject domain domain.

For the case of diagnosis of competencies, dispositions and meta-cognitive skills, xLM can release the evidence supporting the diagnosis. More generally, the history of interaction between learners and LEACTIVEMATH that is used by xLM for diagnosing learner states and dispositions is also available upon request.

3.2 Information interchange

The overall architecture of xLM is depicted in figure 3.1, where the components that closely support xLM diagnostic functionality are shown in yellow (gray). The Extended Learner Model is composed of three main parts: diagnostic, open learner modelling and management. Both the diagnostic components and the management components of xLM are described in this report, although the latter ones are presented here from the perspective of their support to xLM diagnostic functionality. The Open Learner Model, related components and their use of the rest of xLM is described in deliverable D29 (Brna et al., 2005). The arrows connecting distinct components represent information interchange between components. There are two basic types of information interchange, as explained in deliverable D8 (Libbretch 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 program interface (API) for the core diagnostic components of xLM: the Situation 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, though 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 component send their events to xLM Manager for it to distribute them both locally and to the LEACTIVEMATH Event Manager. The exception is the Situation Model, which sends its events directly to the LEACTIVEMATH Event Manager (see below).

- Procedure calls in between xLM components and other LEACTIVEMATH components is indicated in figure 3.1 by the solid arrow on the left of the xLM Manager. Nevertheless, access to LEACTIVEMATH components is

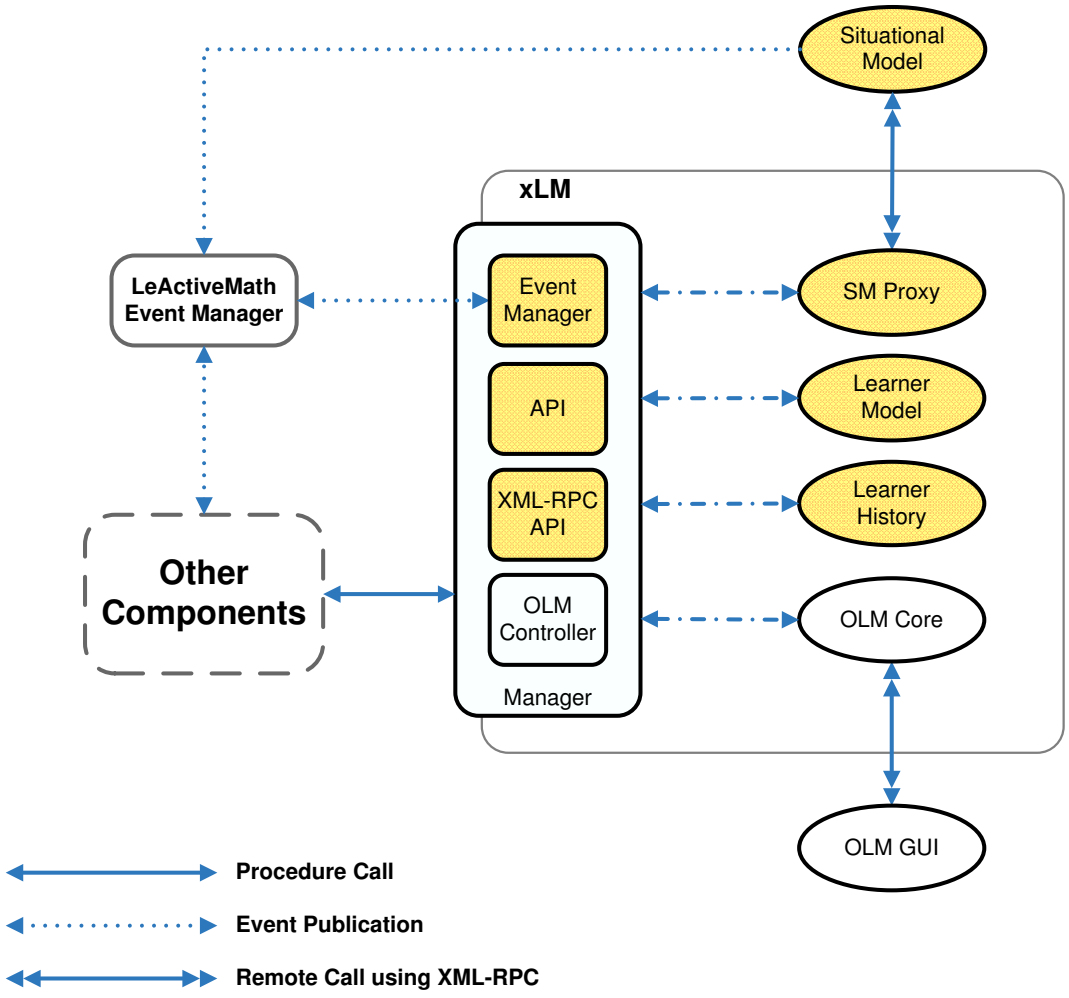


Figure 3.1: Overall architecture of xLM. Solid arrows represent procedure (method) calls and dotted arrows represent event publication. Double-headed arrows stand for remote procedure calls using XML-RPC.

more spread inside xLM than showed in the figure. Details of it will be provided in chapter 4 when relevant for explanation of xLM diagnostic functionality.

- The Situation Model (SM) is executed as an independent process, not necessarily on the same computer where xLM is executed (section 4.2). SM integration into xLM happens through a proxy inside xLM that receives all information and requests send to SM. On the other hand, it became easier for SM to send its events directly to the LEACTIVEMATH Event Manager than to go through its proxy and then the xLM Manager.
- Similarly to SM, 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. 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 via the xLM Manager, as depicted in figure 3.1. Further information interchange via method calls exists between components, sometimes directly and other times indirectly through xLM Manager API.

3.3 Application program interface

The diagnostic functionality of xLM is made available to other LEACTIVEMATH components through its *application program interface* (API), which is built by assembling the API of the SM, LM, OLM and LH. The OLM API is described in deliverable D29 (Brna et al., 2005) and it is not detailed here.

3.3.1 Situation Model

The diagnostic functionality of SM consists basically in the estimation of the learner's needs for autonomy and approval while interacting with LEACTIVEMATH content items (Andrès et al., 2005). Accordingly, SM API is limited to a single method that provides such values encapsulated in an instance of the Face class[1].

```
public Face getFace( String userId);
```

3.3.2 Learner Model

The diagnostic functionality of LM consists in maintaining a model of the learners' qualifications and dispositions towards a subject domain. Therefore, LM

[1] According to Brown and Levinson (1987), autonomy and approval constitute the public *face* of individuals in a society.

provides basic facilities for managing such models, implemented as instances of the class LearnerModel.

```
public void createLearnerModel( String learnerId)
    throws NullPointerException;

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.

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

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 (sections 2.3 and 4.3), otherwise LM responds with a belief standing for complete ignorance (section 4.3.1).

LM beliefs are complex objects, represented as belief functions with numeric representations for certainty, plausibility, ignorance and conflict, plus evidence supporting them. Furthermore, LEACTIVEMATH components are most frequently interested in "summaries" of beliefs, which are in essence decisions on what is most likely the learner case. These are provided also by LM as *summary beliefs*, which still include some measure of uncertainty, or *learner levels*, hard bets with no trace of uncertainty in them.

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

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

Beliefs in xLM learner models include the evidence supporting them. Although it can be requested directly from a belief 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 the item's metadata.

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

public Belief getSummaryBelief( String learnerId, String itemId)
    throws NullPointerException

public Belief getSummaryLevel( String learnerId, String itemId)
    throws NullPointerException
```

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 4.3): Metacog, Affect, Motivation, Competency, CAPEs and Domain.

Differences with D10

The xLM API specified in deliverable D10 includes methods for recovering beliefs and collections of beliefs from learner models by specifying patterns for belief descriptors instead of fully defined descriptors. This methods were included in anticipation to any real need of them, leaving many details to be fulfilled once the need had arisen, but so far it has not. Consequently, this section of the specified xLM API has not been implemented.

Another difference with D10 is the missing implementation of the method

```
List getCAPEs(learnerId,topicId).
```

Formally, it could have been implemented using the already available framework (section 4.3), using the subject domain and CAPEs maps. In practice, to implement it has been very difficult given the fact that misconceptions and common errors identified by authors have not been included in LEACTIVEMATH content until very recently. This issues is further discussed in section 5.

3.3.3 Learner History

The support provided by LH to xLM diagnostic functionality consists in recording all events relevant for learner diagnosis and delivering them on request. 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 or not a learner has already seen a content item.

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

Chapter 4

Implementation

4.1 Introduction

This chapter presents the details of the design and implementation of xLM and its individual components that build up its diagnostic functionality: the Situational Model (SM), the Learner Model (LM) and the Learner History (LH).

As LEACTIVEMATH, most of xLM is implemented using JAVA as programming language, TOMCAT as the web server that hosts and interacts with xLM components, and a collection of JAVA libraries, most of them shared with LEACTIVEMATH. Furthermore, most of xLM runs on the same computer (and JAVA virtual machine) as the rest of LEACTIVEMATH, and it is fully integrated with it. The exceptions to the rule are the Situational Model and the Open Learner Model. The former is implemented in C++ and runs as a separate process, not necessarily in the same computer as LEACTIVEMATH (section 3.2). The latter is implemented in JAVA but its graphical user interface runs as an applet inside the web browser on the user's computer (see Brna et al., 2005).

4.2 Situational Model

The specification detailed two components of the situational model:

1. The situational diagnosis agent (SDA)
2. The situation modeller component (SM-ler)

Whilst both of these two components of SM perform diagnostic functions, they do so at different informational stages. The SDA is responsible for inferring the values of the relevant situational variables based on the information available about the student and their behaviour. The SM-ler, on the other hand, relies on the values inferred by the SDA in order to perform the diagnosis of the situation specified by the SDA in terms of autonomy and approval values. The autonomy and approval values are needed for informing the selection of appropriate tutorial and politeness strategies by the Tutorial Component and the Dialogue Manager respectively. In calculating the values of autonomy and approval, SM-ler necessarily depends on the relationships that exist between the situational

variables and their specific values. Different combinations of these will impact differently on autonomy and approval. For example, low level of student confidence, high aptitude and incorrect answer may lead to diagnosis of a higher autonomy and approval than low confidence, low aptitude and incorrect answer.

4.2.1 Design decisions

The design decisions for the situational model were made on an ongoing basis throughout data analysis. One of the main decisions was made with respect to the input variables to the model. Given the correlations analysis together with the Principal Factor Analysis, we managed to establish a set of necessary variables reducing their number from the original 13 to 7. This was a crucial decisions which was to ensure an easier and more perspicuous implementation of the system. It was also to ensure the feasibility of the diagnosis of the variables' values based on information provided by other sub-components of LEACTIVEMATH. The set of input variables to the situation model was determined as follows:

1. student's confidence
2. correctness of student's answer
3. student's aptitude
4. student's interest
5. difficulty of an exercise
6. importance of an exercise
7. student's effort

The diagnosis of the values for each of the variables in the set follows the specification given in the deliverable D10. For ease of reference we provide a reminder of the basis used for diagnosing the values of the situational variables.

Thus, there are two types of variables modelled by the situational model: those of which the values can be obtained from the metadata associated with concepts or exercises stored in LEACTIVEMATH, and those of which the values need to be diagnosed based on the information provided by other components of the system. The values of the factors *correctness of the student answer*, *difficulty of material* and *importance of material* can be obtained from either the metadata associated with an exercise or from the scenario type. Situational factors for which the values cannot be obtained in this way include *student confidence*, *student interest*, *student aptitude* and *student effort*. These are diagnosed by the situation diagnosis agent. Based on the data analysis , we established seven main sources of evidence which tutors tend to use to infer the values of the four situational factors:

1. *Hesitation level* which is established based on two variables:
   - the elapsed time between the submission of tutor question or instruction and commencement of student response.
   - the expected time for the commencement of a student response which corresponds to the average response time established for that student.
2. *Linguistic cues* of which the specific instances are:
   - use of interrogative forms in student answers (e.g. "?", "...").
   - use of hedges, e.g. "maybe".
3. *Achievement level* of which the estimation depends on three variables:
   - a number of recent student exercises (or steps in an exercise) under consideration (currently, based on our study analysis this number is set to 4).
   - degree of correctness (mark) for each exercise (or step in an exercise).
   - adjacency of the same marks with the number of exercises or steps considered.
4. *Difficulty of material* of which the variable is the rating of difficulty of a particular exercise obtained from the metadata associated with the current and the previous exercise.
5. *Spontaneous admissions* which is established based on presence or absence in student's response of terms that refer their current motivational states, e.g. statements of enjoyment, confusion, boredom, enthusiasm. The presence or absence of specific terms is determined against a look up table established based on the corpus collected in our studies.
6. *Granularity of solution steps* which is determined by comparing two variables:
   - the number of steps taken by the student to present the solution.
   - the number of steps represented in the correct path of the domain reasoner.
7. *Student's initiative* of which the specific (optional) instances include:
   - Student asks a clarification question.

Whilst a number of sources of evidence can be obtained without natural language facilities, some sources are inherently dependent on natural language.

The latter type includes the linguistic cues, spontaneous admissions and student initiative. Most of the seven sources of evidence can be obtained in LEACTIVEMATH without any reliance on natural language dialogue. These include hesitation level, achievement level, difficulty of material and granularity of solution steps. The description of the prototype implementation presented in this document does not include a facility for making inference based on the natural language evidence. Such facility is currently unavailable.

The same source of evidence may be used to contribute to the diagnosis of more than one factor. However, as shown in figure 4.1, the diagnosis of the individual factors relies on a unique combination of the sources of evidence. Additionally, our data analysis suggests that not all sources of evidence are considered to be equally important by the tutors. Some sources of evidence generally contribute more to the diagnosis of certain factors than other ones. This means that each source of evidence may be assigned a weight reflecting its *relative importance* for a particular diagnosis. In table 4.1 we provide a summary of the four factors for which the values need to be determined by the SDA, the specific sources of evidence that contribute to the diagnosis of each of the factors, and the weights that represent tutors' comments about the relative importance of each type of evidence to the diagnosis of a value for a specific factor. The weights are given on a scale from 1 to 5 where 1 means *low* and 5 means *high*.

In addition to the weights representing the relative importance of a source of evidence to the diagnosis of a value of a situational variable, our study also suggests that tutors take into account the *frequency* with which a particular type of evidence occurs in an interaction. For example, hesitation evidence is taken more seriously by the tutors if it is observed several times in the same interaction and recently in relation to the current task, rather than if it occurs only once or if its occurrence is not recent.

Ultimately, the *overall importance* of each source of evidence is calculated based on a *strength value* which is determined by the frequency of occurrence within a current session and on a *relative importance value* which indicates the relevance of a source of evidence to a situational variable. As is described in section 4.2, frequency of occurrence of a particular evidence is established dynamically and cumulatively during a learner interacting with LEACTIVEMATH based on the total number of occurrences of a particular type of evidence in a given interaction and on the recency of such occurrence.

The most significant design decisions were made with respect to the shape and functioning of the situational modeller. As will be discussed in section 4.2, the burden of the calculations of autonomy and approval values, based on the input values supplied to the situational modeller by the situational diagnosis agent, was put on the Bayesian Network (BN) representation of the process shown in figure 4.1. As will be also discussed in more detail in section 4.2, the main reason for this was to secure representational and computational efficiency. To make the effort of implementing the BN representation worth while, our design had

Table 4.1: Importance of Evidence; 1 means *low*, 5 means *high*

FACTOR	SOURCE OF EVIDENCE	WEIGHT
Student Confidence:		
	Student hesitation	5
	Linguistic cues	5
	Spontaneous admissions	5
	Student initiative	4
	Granularity of solution steps	3
Student Interest:		
	Student initiative	5
	Spontaneous admissions	5
	Granularity of solution steps	4
	Achievement	3
Student Effort:		
	Student initiative	5
	Granularity of solution steps	5
	Difficulty of material	4
	Achievement	3
Student Aptitude:		
	Achievement	5
	Difficulty of material	5

to meet certain criteria: (1) conciseness of the set of input variables, (2) clearly defined relationships and a hierarchy of dependencies between input variables and (3) a source and a mechanism for defining conditional probabilities of the represented nodes.

The results of the data analysis allowed us to satisfy all three criteria. With respect to the first criterion we already discussed the method for determining the set of necessary input variables. The resulting seven situational variables represent the input to the situational modeller. In terms of BN representation this means that they occupy the top-most, input position of the network. Following Porayska-Pomsta (2003) and the comments made by our tutors throughout verbal protocols, the representation of the relationships between the input variables assumes that all situational variables modelled affect how a given situation is judged in terms of the autonomy (guidance) and approval required by a particular student in that situation. This is reflected in the fact that all input nodes

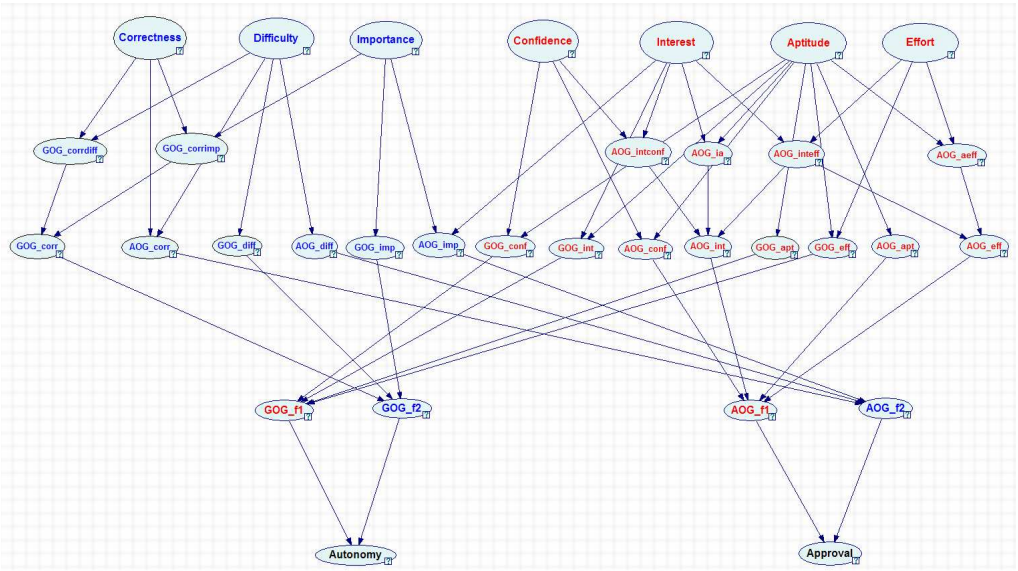


Figure 4.1: The design of the Bayesian network for LEACTIVEMATH (GOG = Guidance Oriented Goals, AOG = Approval Oriented Goals; f1 = Factor 1, f2 = Factor 2.

contribute to the calculation of autonomy and approval represented by the nodes at the bottom-most level of the network. However, the way in which the nodes are combined at the top of the network reflects the especially strong relationships that exist between some, but not all, of the input variables as was determined by the Principle Component Analysis.

The reason for combining situational variables is to yield child nodes that will provide a bridge between the raw input variables such as correctness of student answer or student confidence and the autonomy and approval values. According to Brown and Levinson's version of the Politeness theory, autonomy and approval values represent points on two socio-psychological dimensions relevant to all people. These dimensions refer to people's needs to be independent in their actions whilst at the same time being approved of by others. When put in the context of education, these two dimensions can be interpreted as the students' universal need to be given the freedom of initiative and the freedom to explore the knowledge by themselves, while also obtaining approval for their efforts from the tutors (Porayska-Pomsta, 2003). Because it is usually a tutor's objective to strike a balance between granting the needed freedom to the students without causing them to flounder and to get frustrated, and because it is typically their goal to provide appropriate praise without causing the students to become either over-confident or to loose confidence altogether, an interpretation of autonomy and approval as tutor goals can be afforded. In terms of the design of the situational modeller then, the trick is to find a plausible mapping from the input variables to the goals of providing the student with an appropriate *degree* of autonomy and approval. This degree is to be expressed in terms of the probability of high and low autonomy, and the probability of high and low approval

being required in a specific situation.

In order to map between the situational variables and the autonomy and approval it is necessary to interpret them in terms of *autonomy oriented goals* (GOG, i.e. to guide or not to guide the student) and *approval oriented goals* (AOG, i.e. to approve or not to approve explicitly of the student). A small minority of the situational variables lead to such interpretation directly. For example, based on tutors' comments and actions, if the importance or difficulty of material is high, tutor goal is typically to provide more support in terms of guidance and hints. This means that the autonomy given may be reduced in such a case. However a vast majority of the variables modelled in the situational model need to be combined with other variables in order to be interpreted meaningfully in terms of autonomy and approval oriented goals. For example, in order to determine the appropriate course of action based on the correctness of student answer, it may be necessary to put it in the context of the variables such as difficulty and importance of material respectively. When such combinations are attempted, basic rules for combining the variables emerge, providing a way in which to populate the conditional probability tables. It is the purpose of the qualitative analysis to provide validation and to determine the post-conditions of such rules. The description for the designing principles for the rules used to calculate the conditional probabilities is given in the deliverable D10. However, the results of our further analysis of the data informed the rules specified in D10 by the weights with which each situational variable contributes to the post-condition of a rule. As was also specified in D10, it is the weights and the numerically expressed value of a variable that determine the outcome of combining two or more variables. For example, if the student answer is incorrect (val1), and the difficulty of material is very difficult(val2), then the approval oriented goal based on difficulty might be to provide the student with a sufficient amount of approval in order to prevent the student from being discouraged. The level of approval is calculated by combining val1 and val2 along with their respective weights (W1 and W2) and by dividing them by 2. We used the results of the frequency analysis to determine the weights of the individual variables. The weights represent the overall relative importance of a variable to tutor feedback. In the case of our example then, we can determine W1 to be approximately 0.59, whilst that of W2 to be around 0.84. The weights' values refer to the relative importance of the contributing factors.

In some cases, in order to obtain the desired interpretation of the situational variables in terms of tutor goals it is necessary to combine them with more than one other variable. For example, the interpretation of the correctness of student answer in terms of its guidance oriented goal involves both difficulty and importance of material (the second level in the network). The two interpretations are then combined to yield an integrated guidance oriented goal evoked by the correctness of student answer variables. By the third level, such interpretations are available for all input variables. In cases where the interpretation of a variable depends on its combining with more than one factor, the weights are assigned to

the outcomes of each combination. Continuing with our example, a weight will be assigned to the outcome of the combination of correctness of student answer and difficulty of material, and another weight – to the correctness of student answer and importance of material. These weights are also inferred based on the results of the data analysis – this time on the strength of the correlations between the relevant variables. In the case of correctness of student answer and difficulty of material the weight is .364 and in the case of correctness and importance of material, the weight is .683. As before, these weights along with the numerically expressed values of the variables in question are used to calculate the conditional probabilities which populate the tables of the corresponding nodes in the network.

The fourth level of the network represents the two groups of factors determined by the Principle Factor Analysis. Whilst all variables in each group contribute to both guidance and approval oriented goals, not both groups contribute equally to the calculation of autonomy and approval oriented goals. The conditional probabilities of these nodes are calculated based on rules such as the ones used for populating the nodes at the higher levels. The only difference is the source of the weights associated with each input to a node, which in this case, represents the loading of each variable onto a particular group. This interpretation of the variable loadings was motivated by the results of the data analysis. In particular, our examination of the variables in both groups in conjunction with tutors' interviews and their verbal protocols, resulted in the design decision to associate group 1 with issues of student motivation and group 2 with pedagogical content and student performance. To reflect this difference of reference in our design, we used the loading of every variable onto each group as a measure of the its contribution to the calculation of the autonomy and approval oriented goals. Thus whilst the respective contributions of student confidence, student interest, student aptitude and effort contribute more significantly to the overall value of the approval oriented goal, difficulty of material, importance of material and correctness of student answer contribute less to that goal. Conversely, the contributions of the last three variables to the value of guidance oriented goal are greater than of the former four variables.

The final level in our network design is the one comprising of the autonomy and approval nodes. The same method as described earlier was used to populate these two nodes with the exception of the source of the weight values of the contributing states from the nodes at level four. In this case the weights represent the significance of each of the groups obtained from the Principle Factor Analysis, with the first group being more significant than the second group by virtue of it accounting for more variance in the analysed data.

4.2.2 Implementation

The implementation of the situational model follows closely its design as specification. Three components have been implemented:

1. The situational diagnosis agent responsible for processing the information obtained from other relevant components of the LEACTIVEMATH.
2. The situational modeller used for inferring autonomy and approval values.
3. The system of rules expressing the dependencies and their nature between the situational variables modelled. The rules were used for populating the conditional probability tables in the Bayesian network implementation of the situational modeller.

The system of rules for populating the BN with conditional probabilities is a stand alone system which neither relies on information from LEACTIVEMATH nor does it contribute to it directly.

The implementation of the situational diagnosis agent and of the situational modeller is integrated into one system called the situational model.

The situational model is implemented in C++. The choice of the programming language was dictated primarily by our choice of the software package for building and manipulating the Bayesian network. The implementation of the situational model follows strictly the principles of object oriented programming as shown in figure 4.2. It consists of 14 classes, 10 of which are directly concerned with the situational diagnosis agent and LEACTIVEMATH event handling, 1 - with managing the Bayesian network, 2 - with style and data management, and 1 class is concerned with managing the event traffic between the situational model and LEACTIVEMATH. The last class enables integration of the situational model with LEACTIVEMATH.

Figure 4.2 represents the implementation of the Situational Model including the classes which enable the communication between the Situational Model and the rest of the LEACTIVEMATH system. The XmlRpcServerMethods class is implemented by the LEACTIVEMATH events.

The implementation of the situational diagnosis agent

In deliverable D10 and in chapter 2 we described the basis on which the situational diagnosis agent operates. Specifically we discussed the different sources of evidence needed for inferring the values of student confidence, interest, effort and aptitude. In the current implementation of the Situational Model prototype, four out of the total seven sources of evidence are accounted for. These include the difficulty of material, student hesitation, student achievement and granularity of solution step. The three sources of evidence which are not as yet accounted for are those which rely on the natural language facilities of the system to be operational. Hooks will be provided for these to be added at a later point.

Some of the sources of evidence that are accounted for by the current implementation are provided directly by other components of LEACTIVEMATH. These include the difficulty of material and granularity of solution step. Other sources

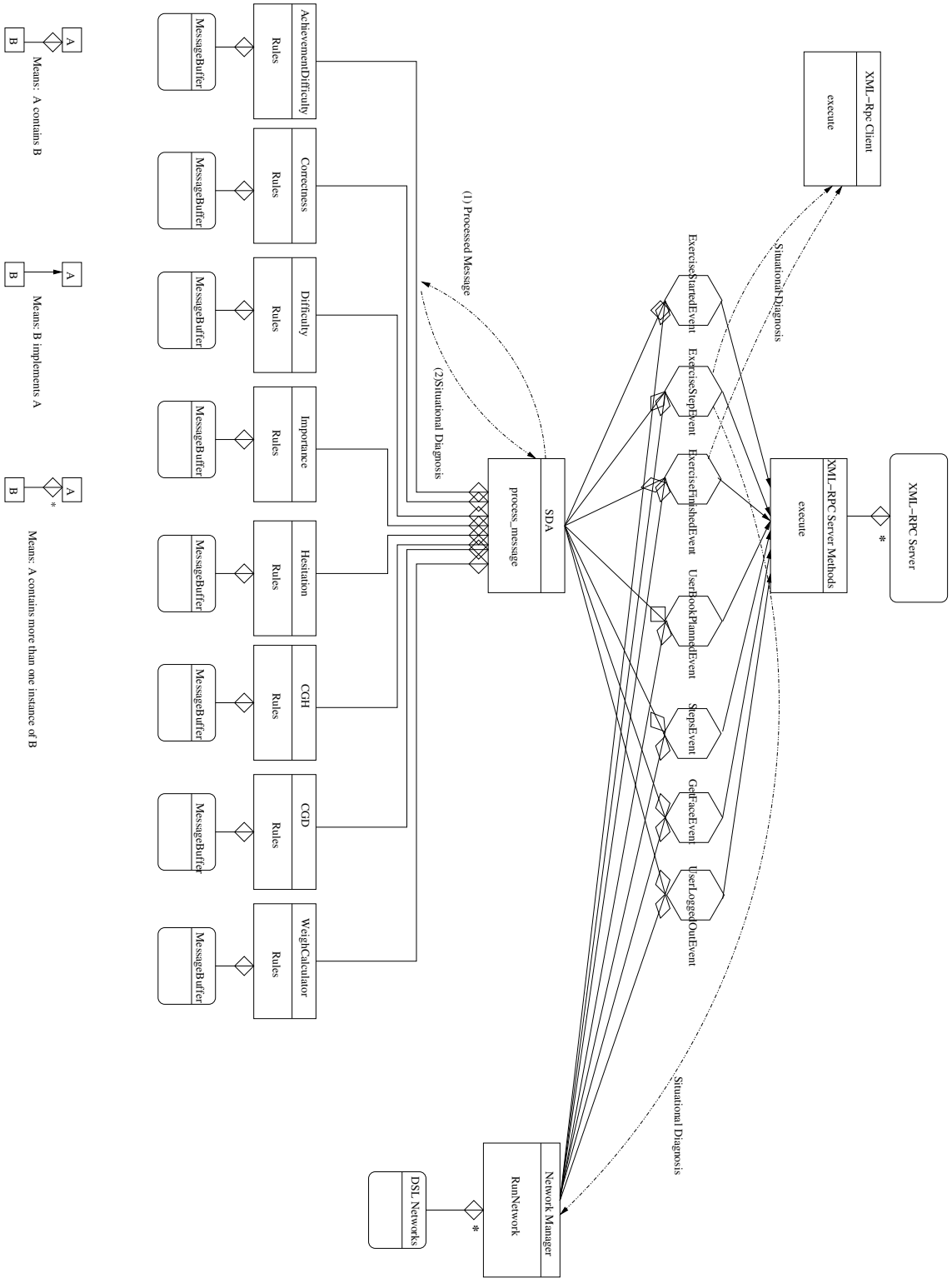


Figure 4.2: Architecture of the implementation of the Situational Model

of evidence need to be inferred based on the relevant information from the rest of the system and sometimes on the information as to the other sources of evidence. The dependent sources of evidence include student hesitation and student achievement. In any case all information that is received from outside the Situational Model, including the information that is not used as evidence for diagnosing situations, is provided in the form of an LEACTIVEMATH event. Event handlers for the appropriate events receive the events and pass them to the relevant classes elsewhere in the system. In sum the following LEACTIVEMATH event handlers are registered in the Situational Model:

- ExerciseStartedEvent, which informs SM that an new exercise was started. It is used to provide information about the *difficulty* of the material. It also creates a copy of a Bayesian network for every learner for which such copy does not already exist. The event is used to infer the value of student hesitation.

- ExerciseFinishedEvent, which is used to infer the composites of the values of student achievement, student aptitude, interest and effort. It is one of the events that triggers the diagnosis of the situational variables to be sent to the xLM. It carries the information about the *success rate* of the solutions provided by the user. It is used to infer the composites of the values of student achievement, student aptitude, interest and effort.

- ExerciseStepEvent, which provides information as to the *difficulty of an exercise* and the *success rate*. Just like the ExerciseFinishedEvent, it is an event that triggers the diagnosed values of the situational variables to be sent to the xLM. It is used to infer the values of student aptitude, interest, effort and achievement.

- UserBookPlannedEvent,which provides information as to the importance of material. It is used only by the Importance object which sets the importance value to *high* if the *Scenario* type input value from the event is ExamSimulation.

- UserLoggedOutEvent, which is used to inform the Situational Model about the fact that the system is no longer in use by a specific user. This event is 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.

Once an event is received from LEACTIVEMATH, the information that it contains is sent to the relevant class for processing. The processing of the information is carried out using methods and situational diagnosis rules specified in detail in the deliverable D10. For each student in a given interaction (identified by the UserID and ExerciseStartedID), the information contained in an ACTIVEMATH event is stored in a *message buffer*. This facilitates easy access to the objects that

need to use such information to fire situational diagnosis rules. For example, information about past events is needed to calculate the average time that a particular student typically spends on exercises. To provide a meaningful estimate of a current level of student hesitation at a particular point in an interaction, it is important to do the calculations on an ongoing basis and with reference to all past events relevant to that interaction. Similarly, access to past events are needed by the AchievementDifficulty class and by the WeightCalculator class. For example, as was specified in the deliverable D10, the rules for infering student achievement, student aptitude, interest and effort rely on the information about correctness of student answer and on the difficulty of a given exercise. They also rely on the information about up to 4 past instances of correct, incorrect and partially correct student answer, and on the adjacency information, (e.g. whether the past incorrect answer events occurred one after another). Finally, the WeighCalculator class uses the same facility to calculate the frequency of a particular source of evidence occurring by storing information on all occurrences of a source of evidence during a given interaction. As was explained in D10, the frequency is one of the two values associated with a source of evidence which contribute to the estimation of the total weight of the value of a situational variables being diagnosed. In summary, message buffers facilitate making the relevant calculations, while by the same token they cut down on the overall processing time by eliminating the need for SM to communicate with LEACTIVEMATH every time the relevant information is required by one of its objects. Message buffers are objects created at run-time and for every needy class and are deleted as soon as the interaction ends. Such maintenance ensure the overall efficiency of the system.

Once the situational diagnosis is available and the sendFactorsToLeAM method instantiates the XML-Rpc client, it sends the information about the values of the situational variables diagnosed by calling the execute method. The situational diagnosis is also sent to the situational modeller which, takes over the diagnosis of a given situation by calculating the autonomy and approval values for that situation.

The implementation of the situational modeller

For the Bayesian network the SMILE[1] library was used which for certain platforms, such as Microsoft Windows and Linux, comes with a graphical user interface (GUI). The GUI (called GENIE) allows one to create the structure of Bayesian networks without having to code them explicitly in C++ which, in the case of the network presented in this chapter, would have been very cumbersome. Once the structure of the network is in place, SMILE functions can be used to create code which reads the network, writes the prior and conditional probabilities into the individual nodes, sets evidence on the nodes and to run the network.

[1]Both SMILE and GENIE are decision-theoretic software and have been developed by the Decision Systems Laboratory, at the University of Pittsburgh (DSL, 1999).

There are two main reasons why we chose the Bayesian network formalism. First, Bayesian networks provide an efficient method for making decisions based on some evidence from the real world. In that sense they provide an intuitively natural way of reproducing certain diagnostic and decision making capabilities of a human tutor. Since a Bayesian network can be considered as a probabilistic expert system (e.g. Guo and Hsu, 2003), this ability is particularly suitable for modelling tutors' expertise in making judgements as to the characteristics of an optimal response in a given situation. More importantly Bayesian networks allow one to model complex dependencies between random variables in an efficient manner. It is precisely the purpose of the current model to represent such complex dependencies between situational variables and to use them to infer the autonomy and approval values. In contrast with rule-based systems, for instance, in which representing such dependencies would not only be very cumbersome, but also very difficult to inspect and to change, Bayesian networks provide a compact way of describing the entire distribution of the variables in terms of manageable and inspectable probability tables (e.g. (Heckermann, 1996); (Guo and Hsu, 2003))—they facilitate efficient storage of data.

Second, Bayesian networks have the ability to instantiate arbitrary subsets of variables (regardless of whether or not they are fully specified or under-specified) and to calculate the conditional distributions on another subset in order to make a decision based on those distributions. In other words, just as they provide an efficient way of storing large amounts of complex data, Bayesian networks have the capability of performing inference in an efficient manner which is ideally suited to the current problem. In the case of the situational modeller, the conditional distributions of all the relevant stages in the process of making a decision about the appropriate levels of autonomy and approval are based on the instantiations of the situational variables provided as the input by the situational diagnosis agent.

Given a representation of the Bayesian network in terms of a Genie graphical representation, we relied on the C++ SMILE library to manipulate the network. The main class which implements the methods for network manipulation is the NetworkManager class. It is to this class that the diagnosis from the SDA is sent for use as evidence to the network's top-most nodes. The nodes represent the situational variables modelled in LEACTIVEMATH.

Every node in a Bayesian network consists of at least two *states*. When evidence is set on a node, only one state is flipped to TRUE, while the rest of the states are flipped to FALSE. Specifically in relation to the current model, this is very suitable for representing the mutual exclusiveness of situational variable-values in the input nodes. In any given situation, the variable *student's confidence*, for example, may have a value *very high confidence* or *high confidence*, *medium confidence*, etc., but nor more than one of the possible values simultaneously. In this case, *student confidence* represents the node, while *very high confidence* and *high confidence*, etc., represent two of its possible output states.

In the LEACTIVEMATH network, the nodes consist of conditional probability tables, whereby there are up to 5 output states depending on the actual variable represented by a given node. The states for all the top-level nodes in the network represent the possible values of the situational variables modelled.

Every node in the network carries information in the form of a probability table, which expresses the probability of a node being in one of the pre-specified states. If a node represents pre-conditions only, its definition will consist of a number of fully specified possible states, along with their marginal probabilities. For example, if a node A with the states *a1* and *a2*, does not have any parents, its probability table will merely express the respective probabilities of *a1* and *a2* (see figure 4.3). In our network, all of the top-most nodes represent preconditions. The marginal probabilities associated with each output state in such a node are the same for each state and when summed they add up to 1. This is to reflect the fact that *a priori* any such state has an equal chance of being observed. When such observation occurs, evidence is set on the node, and the appropriate state is selected. The value of such a state is therefore flipped to 1 and the values of the remaining states are flipped to 0. If nodes are connected, the output states of the parent nodes constitute the input states to the children nodes.

If a node represents a rule, the definition will consists of a number of its possible states, all of the combinations of all the states of that node's pre-conditions (i.e. the parent nodes) and prior, conditional probabilities for every such combination. For example, if a node C with the possible states *c1* and *c2* is a child of A and B with the states *a1*, *a2*, and *b1*, *b2* respectively, then the probability table for node B will express the respective probabilities of *c1* and *c2* given *a1* and *b1*, then given *a1* and *b2*, then *a2* and *b1*, and finally *a2* and *b2*. This is illustrated in figure 4.3. The calculation of the marginal probability of the state *c1* would take the form of the equation in 4.1.

$$\begin{aligned} P(c1) = &(P(PrA1) * P(PrB1) * P(c1|PrA1, PrB1)) \\ &+(P(PrA1) * P(PrB2) * P(c1|PrA1, PrB2)) \\ &+(P(PrA2) * P(PrB1) * P(c1|PrA2, PrB1)) \\ &+(P(PrA2) * P(PrB2) * P(c1|PrA2, PrB2)). \end{aligned} \tag{4.1}$$

Whilst, theoretically, there is no limit on the number of possible states that a node may have, an increase in the number of possible states will lead to an increase in the complexity of the definitions of its children nodes which may also increase the complexity of the inference needed to propagate the values through a network. The complexity of the definitions of children nodes increases exponentially if a child node depends on two or more parent nodes. This is why in the LEACTIVEMATH network we allow only four parents per node. This also ensures that our rules for populating the tables are inspectable and easy to modify, should the need for such an inspection and modification arise.

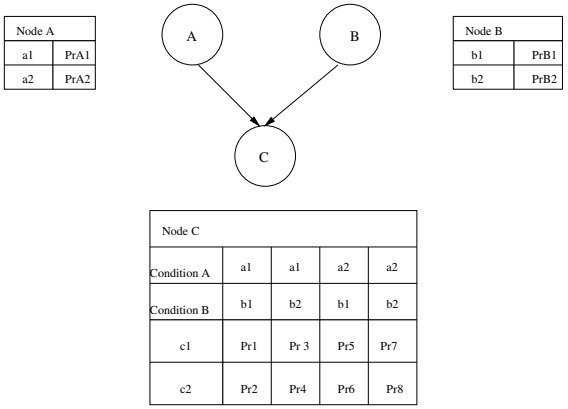


Figure 4.3: An example of a simple three-node network.

The running of the network happens after the appropriate evidence is supplied. Running the network means propagating the evidence through the network by means of an appropriate probabilistic inference algorithm.

There are several inference algorithms available to the designers of Bayesian networks. The most commonly used is the type of algorithm proposed by (Lauritzen and Spiegelhalter, 1988) and further developed by (Jensen, 1996) amongst others. In this type of algorithm a Bayesian network is transformed into a tree in which each node corresponds to a subset of variables in the transformed network. The transformation into a tree is sometimes necessary to avoid an infinite loop in the propagation of values across the nodes. Such a problem can arise in cases where in the Bayesian network there exists an acyclic loop between the nodes. The choice of an algorithm typically depends on the size of a given network. For larger networks it is best to use an algorithm which exploits stochastic sampling methods such as those proposed by (Henrion, 1988), (Shachter and Peot, 1990) or (Fung and Del Favero, 1994). The Lauritzen-Spiegelhalter algorithm is suitable primarily for relatively small-sized networks (DSL, 1999). This is also the algorithm used in the current implementation.

Once the evidence is set and propagated through the network, the values of the output states of its bottom-most nodes can be inspected. These nodes represent the autonomy and approval dimensions, whilst the values of their output states represent the likelihood of their occurrence given the input situation. In the case of autonomy and approval nodes, these states are *autonomy*, *no autonomy*, and *approval*, *no approval* respectively. The autonomy and approval values sent to LEACTIVEMATH will always be those corresponding to the output states *autonomy* and *approval*.

An important feature of the NetworkManager is the fact that it allows for multiple copies of the network to be maintained simultaneously. This is necessary to enable modelling of the immediate situations for a number of users at the same time, thereby conforming to the requirements of the overall system. When a new user logs in to LEACTIVEMATH, a copy of the uninitialised network is created

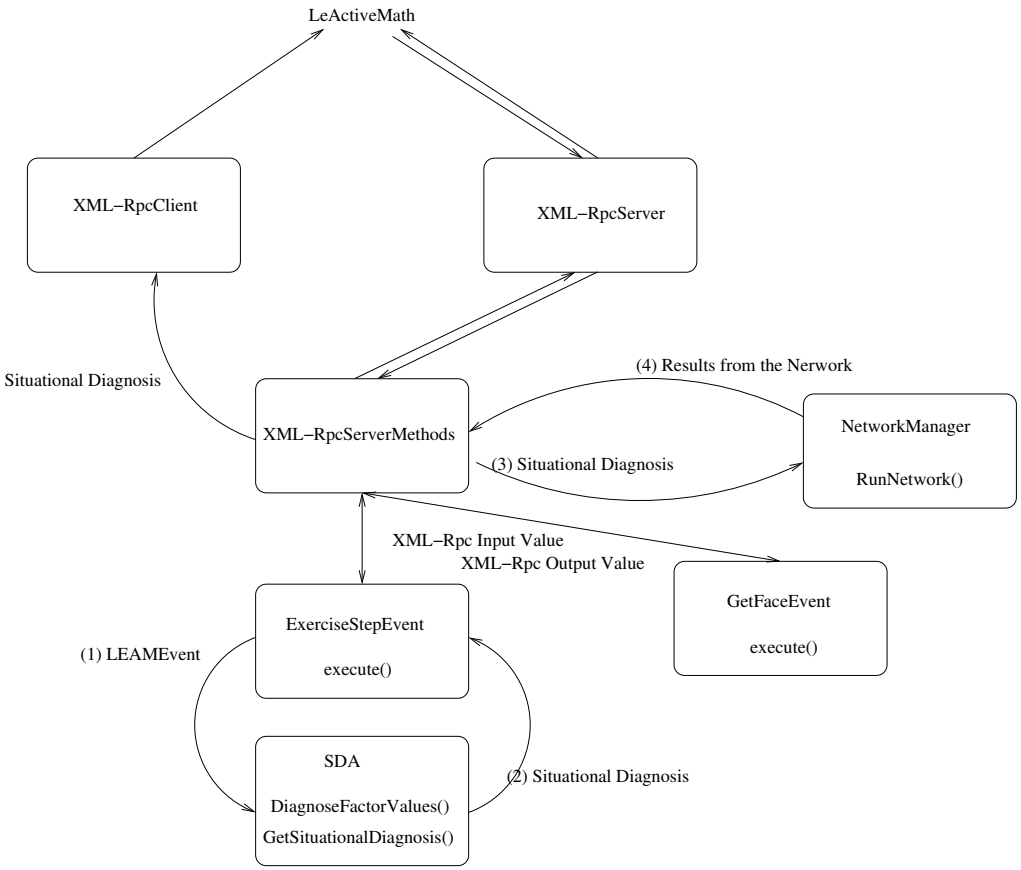


Figure 4.4: The overall path from the point an event is received from LEACTIVEMATH to the point at which the output of SM is provided.

for them and is kept private to their session. The copy of the network is deleted at the end of every interaction, at the point when the situational model receives the UserLoggedOutEvent from LEACTIVEMATH.

4.2.3 The flow of information within the Situational Model

Given the overview of the system implementation architecture the path from the point at which an event is received from LEACTIVEMATH to the output values of autonomy and approval is illustrated in figure 4.4. As was explained earlier, the LEACTIVEMATH events constitute the basis for all diagnosis done by the Situational Diagnosis Agent (SDA).

The event handling classes (for example ExerciseStepEvent) all implement XmlRpcServerMethod which allows them to be registered with the XmlRpcServer object created at the start of program execution. The XmlRpcServer communicates with the XlmEventManager. This server object binds itself to a port (currently 27777) and listens for incoming traffic from XlmEventManager. When a message is received from the XlmEventManager the server works out which

of the XmlRpcServerMethod objects it contains. These will be one of ExerciseStartedEvent, ExerciseFinishedEvent, ExerciseStepEvent, UserBookPlannedEvent and UserLiggedOutEvent. It then handles the particular message and calls its execute method, passing the message as an input parameter to the Situational Model's situational diagnosis agent (SDA). In the case of the example illustrated in the figure the event received is the ExerciseStepEvent.

The event handlers execute methods translate the incoming message from an XmlRpcValue to an AMEvent, which then gets passed into SDA for processing. This is where all the Situational Model's internal processes take place. These involves calling DiagnoseFactorValues and GetSituationalDiagnosis methods on SDA. ExerciseStepEvent carries the information as to the *timestamp* of the event, *userID*, *exerciseID*, the *difficulty* of the exercise and the *success rate* of the solution provided by the learner. This information is used by the SDA which manages all the classes that are responsible for performing situational diagnoses. Specifically, the *timestamp* information is used by the Hesitation class which is responsible for diagnosis the level of student hesitation in a manner which has been described earlier.

The *success rate*, which is expressed in terms of numerical values between 0 and 1, is first of all mapped onto the fuzzy-linguistic correctness values *correct*, *incorrect* and *partially correct*. This mapping involves the following rules:

```
(nSuccessRate < 0.3)
    correctness = Incorrect ;
(nSuccessRate < 0.6)
    correctness = Pcorrect;
else
    correctness = Correct;
```

Difficulty and *success rate* are pushed from the SDA class into the Difficulty and correctness classes and to the AchievementDifficulty class respectively. No inference is performed by the correctness and difficulty objects. AchievementDifficulty class contains many inference rules—it is the largest situational diagnoses class. The AchivementDifficulty class is responsible for diagnosing student aptitude, interest, effort and achievement. In turn, student achievement is used by the GCD (GranularityCorrectnessDifficulty) class in conjunction with the granularity of the solution provided by the student to infer student interest.

The situational values diagnosed by the individual classes are passed back to the SDA where they are consolidated through weighted means functions. The consolidate values are the final values which constitute the situational diagnoses ready to be passed on to the xLM and the rest of the LEACTIVEMATH.

The situational diagnosis returned is also passed on to the NetworkManager which is responsible for running the network. The results of running the network are passed back to LEACTIVEMATH via the return parameter of the event handler for GetFaceEvent, a special type of event used by the SM proxy inside

xLM 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.

4.3 Learner Model

In LM, a learner model is a collection of beliefs about the learner's states and dispositions arranged along six dimensions (see figure 4.5): subject domain, competency, motivation, affect, meta-cognition and conceptual and procedural errors (CAPEs). Each of these dimensions is described in a concept map which speci-

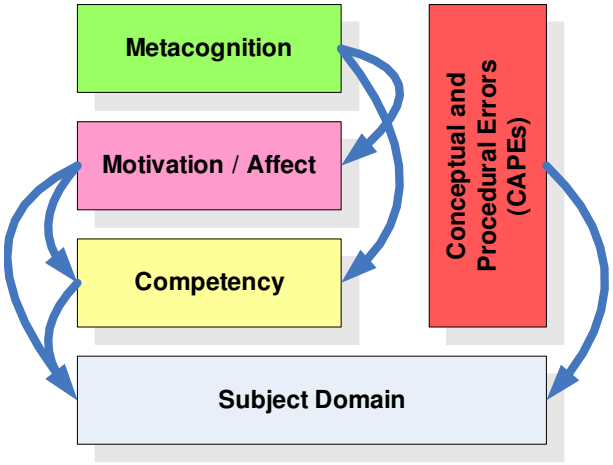


Figure 4.5: A multi-dimensional and layered structure for learner models.

fies the individual factors in the dimension relevant to learning and considered in LM learner models—the maps used in the current implementation of LM are included in appendix A. The maps also specify how distinct factors in each dimension relate to each other. For example, in the current implementation the subject domain is a branch of mathematics known as Differential Calculus and breaks down into domain topics such as 'function', 'derivative' and 'chain rule'; competency is mathematical and decomposes according to the PISA framework (OECD, 2003), and motivation decomposes into factors that are considered to influence learner motivation, such as their interest, confidence and effort put into learning the subject domain and mathematical competencies.

The layered structure of learner models in LM specifies how the modelled dimensions interact with each other. At the bottom of the stack is the subject domain, underlining the fact that learner modelling occurs within a subject domain. Nevertheless, a learner model does not hold any belief about the subject domain *per se*. but beliefs about learner dimensions *applied to* the domain. Hence, on top

of the subject domain are the layers of competency, motivation, affect and meta-cognition, each one relying on the lower layers for expressing a wide range of beliefs about the learner in relation to:

- mathematical competencies on the subject domain (e.g. the learner's level of problem solving with the chain rule);
- motivational dispositions towards the subject domain (e.g. the learner's level of confidence in relation to differential calculus) or towards competencies on the subject domain (e.g. the learner's level of effort in solving problems with derivatives);
- affective dispositions towards the subject domain (e.g. the learner's level of anxiety in relation to functions) or towards competencies on the domain (e.g. the learner's level of satisfaction in relation to solving problems with the chain rule);
- meta-cognitive skills with respect to competencies on the subject domain (e.g. the learner's ability to monitor their competence to solve problems using the chain rule);
- meta-cognitive skills with respect to motivational dispositions towards the subject domain or towards competencies on the domain (e.g. the learner's ability to control their confidence on their competency on differential calculus);
- meta-cognitive skills with respect to affective dispositions towards the domain or towards competencies on the domain (e.g. the learner's ability to monitor their satisfaction with respect to solving problems with derivatives).

Conceptual and procedural errors (CAPEs) is a special dimension, somehow in between competency and the subject domain. CAPEs are not generic, as competencies, but specific to particular domain topics, hence their applications to the domain are predefined. Moreover, neither motivation nor affect nor meta-cognition apply to CAPEs, under the assumption that they are not perceived by learners.

The internal structure of each dimension is defined in its map and is used by LM for specifying the beliefs it can hold, as well as for specifying relations in between beliefs that are used for propagating evidence among them (section 4.3.3). The maps currently in use by LM are included in appendix A.

4.3.1 Levels and beliefs

For simplicity of explanation, let us assume we are talking about a *belief* in a learner model concerning *the learner's competency in posing and solving mathematical problems with derivatives*. Mathematical competencies of learners are measured

in a discrete scale of four levels, from an entry level I to a top level IV, borrowing from the encoding of competency level in content metadata, as described in deliverable D6 (Godauze, 2005). The mathematical competency of a learner is considered *be at* one of these levels, having achieved and passed all previous levels. Consequently, a belief on the learner's competency to pose and solve problems with derivatives is represented in LM as the equivalent to a statement about *which level the learner's competency in posing and solving mathematical problems with derivatives is more likely at*.

Generalising the example above, every belief in LM is about a learner's level on something, as far as that something can be expressed as the application of upper dimensions to lower dimensions in the learner modelling framework (figure 4.5). All dimensions in LM are measured in a similar scale of four levels, from an entry level I to a top level IV.

Belief functions

There are many ways to represent beliefs such as the ones stored in LM. A symbolic representation using a simple logical formalism would look something like

$$\mathrm{B}(\mathrm{solve}(\mathrm{derivative}), \mathrm{II}).$$

A numeric representation using probabilities would assign a probability to each possible level, looking something like

$$\mathrm{B}(\mathrm{solve}(\mathrm{derivative}), p(\mathrm{I}, 0.1), p(\mathrm{II}, 0.60), p(\mathrm{III}, 0.25), p(\mathrm{IV}, 0.05)).$$

In LM, a belief in a learner model is represented and updated using a numeric formalism known as the Transferable Belief Model (TBM) (Smets and Kennes, 1994), a variation of Dempster-Shafer Theory (DST) (Shafer, 1976). TBM and DST use *belief functions* instead of probability distributions. A first difference between a probability distribution and a belief functions is that, while the former assigns a number in the range of $[0, 1]$ to each possible state of the world—e.g. each level the learner's competency could be at—the latter does the same but to each set of possible states of the world. More formally, if we call Θ the set of all possible states of the world,

$$\Theta = \{\mathrm{I}, \mathrm{II}, \mathrm{III}, \mathrm{IV}\}, \tag{4.2}$$

then a probability distribution is a function p from Θ to $[0, 1]$, whereas a belief function b goes from the set of all sets of levels in Θ to $[0, 1]$,

$$b : 2^{\Theta} \rightarrow [0, 1]. \tag{4.3}$$

Mass, belief and plausibility A belief in a learner model can be represented at least in three different ways, as a *mass*, a *certainty* or a *plausibility* function, three

types of belief functions[2]. Although they are equivalent, a mass function is the easiest to manipulate computationally and hence is the one used in LM.

If s_0 is a set of levels, say $s_0 = \{\text{III}, \text{IV}\}$, the mass of s_0, or $m(s_0)$ can be interpreted either subjectively or objectively.

- *Objectively*, as the support the evidence gives to the case that the true competency level of the learner is in the set s_0 (i.e. it is either III or IV) without making any distinction between the elements of the set[3].
- *Subjectively*, as the part of the belief that pertains exclusively to the likelihood that the true learner's competency level is in s_0, without being any more specific towards either of the levels.

A mass function in LM is generally required to satisfy the requirement that the sum of all its values must be one; i.e.

$$m : 2^{\Theta} \rightarrow [0, 1] \quad \text{such that} \quad \sum_{s \in 2^{\Theta}} m(s) = 1. \tag{4.4}$$

However—as in TBM and differently from DST—the mass assigned to the empty set does not have to be zero, and it is interpreted in LM as the amount of *conflict* in the evidence accumulated[4]. The mass assigned to the set of all possible levels Θ is generally interpreted as the amount of complete *ignorance* in a belief.

4.3.2 Evidence

Evidence for learner modelling comes into LM in the shape of *events*, mostly representing what has happened in the learners interaction with educational material and the rest of LEACTIVEMATH. Some events comes from inside xLM, as is the case for events generated by the Situational Model and Open Learner Model.

Events are raw evidence that needs to be *interpreted* in order to produce mass functions that can be incorporated into beliefs in learner models using a *combination rule* (Sentz and Ferson, 2002). Currently, two categories of events are interpreted by LM and their evidence incorporated into beliefs: *behavioural* events and *diagnostic* events. Events in the first category simply report what the learner has done or achieved, whereas events in the second category report a judgement of learner levels produced by some diagnostic component of LEACTIVEMATH.

[2]We call *certainty function* what is more more commonly known as a (proper) *belief function*. We do this to avoid the confusions between LM beliefs, generic belief functions and a particular type of belief function.

[3]Imagine Theta is an international tennis team composed by Indian Papus (I), Japanese Takuji (II), Scottish Hamish (III) and French Pierre (IV). They have played against the American Dream Team and you hear on the radio than all of them, except a European one, have lost their matches. If this is all the evidence you have about who in Theta won its match, it certainly supports the case that the player is in the set $\{\text{Hamish}, \text{Pierre}\}$ but does not distinguish between them.

[4]This is supported by the close world assumption that Θ is the set of *all possible levels*.

Interpretation of behavioural events

Given an event of type ExerciseFinished, reporting that a learner has finished an exercise with some *success rate*, LM interprets it to produce evidence for updating the learner model. The resulting evidence would be a collection of mass functions over the following set[5]:

$$\begin{aligned}\Phi = \quad & \{\{\mathrm{I}\},\{\mathrm{II}\},\{\mathrm{III}\},\{\mathrm{IV}\},\{\mathrm{I},\mathrm{II}\},\{\mathrm{II},\mathrm{III}\},\{\mathrm{III},\mathrm{IV}\},\\ & \{\mathrm{I},\mathrm{II},\mathrm{III}\},\{\mathrm{II},\mathrm{III},\mathrm{IV}\},\{\mathrm{I},\mathrm{II},\mathrm{III},\mathrm{IV}\}\}.\end{aligned} \tag{4.5}$$

For the particular case of an event of type ExerciseFinished in the current implementation of LM, these levels are actually *competency levels*. In the general case, the nature of the levels will depend on the nature of the belief the evidence is relevant to—conceptual or procedural error (CAPE), competency, motivation, affect or meta-cognition levels.

In order to make the explanation easier to follow, let us consider a concrete case: interpreting an ExerciseFinished event that resulted from the learner finishing an exercise with metadata as in listing 1. An ExerciseFinished event comes with its own additional information:

- learner identifier,
- exercise identifier,
- session identifier, and
- success rate achieved by the learner in the exercise (in the range $[0, 1]$).

Beliefs addressed By interpreting events of type ExerciseFinished, LM generates direct evidence for beliefs grounded on the subject domain topics the exercise is for, or depends on. For example, the exercise K3_TIMSS is for a learning object with identifier deriv_higher/def_nthdiff, which is mapped to the domain topic *higher_derivative* that represents the abstract notion of higher order derivative. The metadata listed above indicates the exercise depends on the learning object deriv/def_diff_f which is mapped to the topic *derivative* that stands for the abstract notion of derivative. Consequently, all direct evidence produced from an ExerciseFinished event from this exercise will be evidence for beliefs grounded on the topics *derivative* and *second_derivative*.

Metadata indicating which mathematical competencies the exercise trains on, or evaluates, provides further details of which beliefs should be affected by the event. For the example at hand, this are beliefs on competencies generalise (a sub-competency of think mathematically) and decode (a sub-competency of model mathematically)—see map for mathematical competency in appendix A. Metadata on competency level, on the other hand, does not affect the selection

[5]Technically, the domain of a mass function is $2^\Theta = \{s | s \subseteq \Theta\}$. However, given the fact that levels are ranked it makes no sense to have mass for sets that are not intervals, such as $\{\mathrm{I}, \mathrm{III}\}$. In other words, the *focus* of m is always going to be a subset of Φ, which in turn is a subset of 2^Θ.

Listing 1: Example metadata for an exercise.

```
<exercise id="scq_K3_TIMSS" for="deriv_higher/def_nthdiff">
  <metadata>
    <Title xml:lang="en">
      Acceleration of a straight forward movement
    </Title>
    ...
    <extradata>
      <depends-on><ref xref="deriv/def_diff_f"/></depends-on>
      <learningcontext value="secondary_education"/>
      ...
      <field value="physics"/>
      < difficulty value="easy"/>
      <competency value="think" subvalue="generalize"/>
      <competency value="model" subvalue="decode"/>
      <competencylevel value="simple_conceptual"/>
      <typicallearningtime value="00:00:20"/>
      <representation value="verbal"/>
      <abstractness value="neutral"/>
    </extradata>
  </metadata>
  ...
</exercise>
```

of beliefs but the actual production of evidence for the chosen beliefs. Therefore, the beliefs to be directly affected by our example of event would be *beliefs related to the learner's generalisation of results and decoding mathematical models involving first order and higher order derivatives*.

Those beliefs could be directly on the learner's competencies, on their motivational or affective dispositions towards these competencies, on their meta-cognitive skills in relation to these competencies, etc. On the other hand, no belief on a conceptual or procedural error will be directly affected, since events of type ExerciseFinished do not provide any information on CAPEs (CAPEs are reported by events of type ExerciseStep). In the current implementation of LM only evidence for beliefs on learners' mathematical competencies on the subject domain will be produced from events of type ExerciseFinished. The motivational evidence in ExerciseFinished events is extracted by the Situational Model and delivered as SituationFactorChanged events to LM. The affective consequences of finishing the exercise are reported by learners themselves through LEACTIVEMATH Self-Report Tool, which delivers SelfAssessment events to LM. Finally, modelling of learners' meta-cognition by LM depends on OLMMetacog

Table 4.2: Effect of the metadata value for competency level on the estimated difficulty of an exercise for a learner at a given competency level.

Competency level of exercise	Competency level of learner			
	I	II	III	IV
I	-	-	very easy	very easy
II	-	-	-	very easy
III	very difficult	-	-	-
IV	very difficult	very difficult	-	-

events produced by the Open Learner Model component.

Generation of evidence Once the beliefs to be affected by an event have been identified, the next step is to generate the corresponding evidence: a mass function per belief over the sets of levels in Φ. Although most of the metadata of an exercise could have an impact on the evidence to be produced, the production of a piece of evidence from an ExerciseFinished event in the current implementation of LM only takes into account

- the relationship between the exercise and the belief addressed (i.e. whether the exercise is *for* the topic the belief is about, or only *depends on* it),
- the competency level of the exercise,
- the difficulty of the exercise and
- the success rate reported in the event.

The competency level of an exercise is used to determine who should find the exercise either very easy or very difficulty, and who may find it otherwise (i.e. easy, medium or difficult, the remaining grades of difficulty in LEACTIVEMATH vocabulary for metadata on the difficulty of exercises). For example, exercise K3_TIMSS being an *easy* exercise for competency level II (simple conceptual) means it should be a *very easy* exercise for any learner with competency level IV. That would be the case even if the exercise were a very difficult one, given the differences between levels II and IV. Table 4.2 presents an initial estimation of the difficulty of an exercise for a learner, given the competency levels of the exercise and the learner. In other words, the competency level of an exercise sets the shape for a probability assignment function (for being successful in an exercise) outside the critical area surrounding the point with coordinates given by the competency level and difficulty of the exercise (the main diagonal in table 4.2). The metadata for difficulty of the exercise helps to fill the missing cells in table 4.2. A possible interpretation of these metadata is given in table 4.3.

At this stage, knowing the metadata for an exercise such as K3_TIMSS provides estimates of how difficult that exercise would be for learners with different competency levels. For exercise K3_TIMSS the estimates are

Table 4.3: Effect of the metadata value for difficulty on the estimated difficulty of an exercise for a learner at a given competency level.

Exercise		Difficulty of the exercise for a learner at a given competency level			
competency level	difficulty	I	II	III	IV
I	VE	VE	VE	VE	VE
	E	E	VE	VE	VE
	M	M	VE	VE	VE
	D	D	E	VE	VE
	VD	VD	M	VE	VE
II	VE	M	VE	VE	VE
	E	D	E	VE	VE
	M	VD	M	VE	VE
	D	VD	D	E	VE
	VD	VD	VD	M	VE
III	VE	VD	M	VE	VE
	E	VD	D	E	VE
	M	VD	VD	M	VE
	D	VD	VD	D	E
	VD	VD	VD	VD	M
IV	VE	VD	VD	M	VE
	E	VD	VD	D	E
	M	VD	VD	VD	M
	D	VD	VD	VD	D
	VD	VD	VD	VD	VD

Learner's competency level	Difficulty of exercise K3_TIMSS
I	**difficult**
II	**easy**
III	**very easy**
IV	**very easy**

We can see that the exercise does not discriminate between learners with competency level III or higher. Hence, mass should not be assigned to levels III nor IV alone, but only to the set $\{\text{III}, \text{IV}\}$ or sets containing it.

Here is the point when LM needs to translate the qualitative tags denoting difficulty into quantitative measures. In other words, it needs to estimate, for every rate of success r, the probability P of achieving r given difficulty d. It needs to estimate $P(r|d)$. This can be done in many ways, but LM uses a bell-shaped

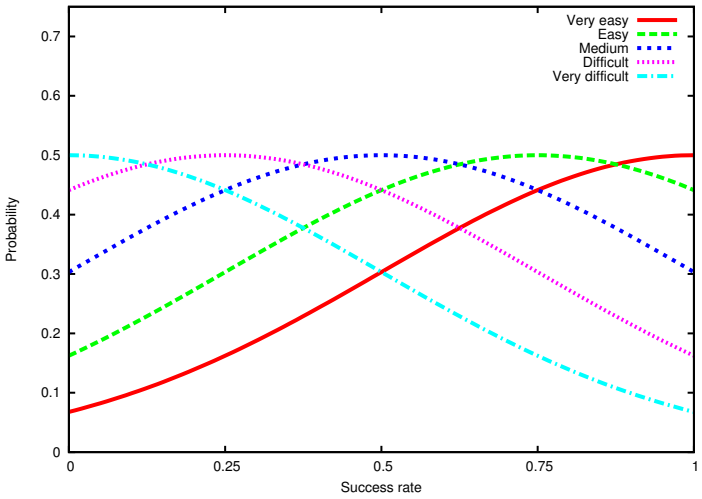


Figure 4.6: Probability assignment functions for each degree of difficulty.

function

$$P(r) = \delta e^{-(r-\mu)^2/2\delta^2}$$

with parameters determined by difficulty as specified in table 4.4. This function gives a 0.5 probability to being completely successful ($r = 1$) in a very easy exercise, and the same probability to being moderately successful ($r = 0.75$) in an easy exercise, just fair ($r = 0.5$) in a medium exercise, unsuccessful ($r = 0.25$) in a difficult exercise and completely unsuccessful ($r = 0$) in a very difficult one. The collection of probability assignment functions for each degree of difficulty is show in figure 4.6.

Table 4.4: Parameters for normal distribution per difficulty value.

Difficulty	Parameters	
	μ	δ
very easy	1	0.5
easy	0.75	0.5
medium	0.5	0.5
difficult	0.25	0.5
very difficult	0	0.5

For example, if the success rate reported in an ExerciseFinished event for exercise K3_TIMSS is $r = 0.8$ then we get the following probabilities per competency level: 0.2730 (I), 0.4975 (II) and 0.4615 (III and IV).

An easy way of translating this probabilities into a mass function would be by normalising the probabilities obtained above and assigning them to the singletons $\{I\}$, $\{II\}$, $\{III\}$ and $\{IV\}$. However, as it was said before, exercise K3_TIMSS

does not distinguish between learners with competency levels III or IV, hence it does not provide evidence for the learner having any of these levels in particular but, in any case, just of having any of them. What should be done if all probabilities above where the same? A possibility, in analogy to the previous case, is to consider the exercise as unable to discriminate between the possible levels of competency of the learner, and hence to interpret it as providing no new evidence at all. Technically, this means the mass distribution in this case should be the one corresponding to *total ignorance* (or complete lack of evidence):

$$m(\Theta) = 1 \text{ and } m(s) = 0 \text{ for all other } s \neq \Theta. \tag{4.6}$$

We can generalise these two cases to an iterative method for calculating a mass functions from probabilities:

1. If there are levels $l_1, l_2, \ldots, l_n$ with non zero probabilities, then take all of them into a set $s = \{l_1, l_2, \ldots, l_n\}$ and make $m(s)$ equal to the smallest probability, i.e. $m(s) = \min(p(l_1), p(l_2), \ldots, p(l_n))$.
2. For every level l_i in s make its probability equal to $p(l_i) - m(s)$. This would reduce to zero the probability of at least one level.
3. Remove from s all levels with re-calculated probability equal to zero and start again at step (1).
4. Finally, scale all $m(s)$, $s \subseteq \Phi$, uniformly, so that the total mass $\sum_{s \subseteq \Phi} m(s) = 1$.

The first step takes the evidence that the event distributes equally among the competency levels it gives evidence of, and puts it into the set of these levels. The second step removes the shared evidence from these levels and the third step removes the levels that have no further evidence supporting them.

The application of this method to the case of exercise K3_TIMSS with success rate of 0.8 and the probabilities calculated above would produce the following mass function:

$$m(\{\text{I, II, III, IV}\}) = 0.549,$$
$$m(\{\text{II, III, IV}\}) = 0.379,$$
$$m(\{\text{II}\}) = 0.072,$$
$$m(s) = 0.0 \text{ for any other } s \subset \Phi.$$

In words, this is weak evidence for the learner being at competency level II (simple-conceptual) and stronger evidence for they being at a competency level in $\{\text{II, III, IV}\}$. However, the mass function includes a fair amount of ignorance that suggest it is still plausible for the learner to be at any competency level, including level I.

Each belief that have to do with topics trained on, or evaluated by the exercise would receive such an evidence. Beliefs concerning topics the exercise depends on would receive a *discounted* evidence with increased ignorance:

$$\begin{gathered} m'(s) = d \times m(s) \text{ for all } s \subseteq \Theta \text{ and then} \\ m'(\Theta) = m(\Theta) + (1 - d), \end{gathered} \tag{4.7}$$

where d is a discount factor in between zero and one.

Interpreting diagnostic events

The interpretation of diagnostic events by LM is simpler than the interpretation of behavioural events given the fact that an estimation of the learner level is included in diagnostic events. Based on how much LM trust the source of the event, the original estimation of the learner level is transformed into a probability distribution over the set of levels Θ. This probability distribution is then transformed into a mass function following the same procedure as explained in section 4.3.2.

Consider, for example, the case of LEACTIVEMATH Self-Report Tool, which is presented to learners every time they complete an exercise so that they can provide feedback on their states of liking, pride and satisfaction—which are assumed to be with respect to the exercise just finished. The values input by the learners are reported to LM in SelfReport events. Then LM transform a single value per factor into a probability distribution by choosing a suitable Beta distribution[6] from the collection shown in figure 4.7.

The mass function resulting from the interpretation of the event would constitute evidence for beliefs on the learner's affective dispositions towards the subject domain topics and competencies that result from considering the exercise metadata, as explained in section 4.3.2.

4.3.3 Propagation of evidence

Interpretation of events such as the ones described in section 4.3.2 provides direct evidences for some beliefs in a learner model. These evidences are propagated to the relevant parts of the learner model following the associations between elements in the maps for each layer in the learner modelling framework (appendix A). The algorithm used for belief propagation is a "loopy" variation of Shenoy-Shafer algorithm for belief-functions propagation (Shenoy and Shafer, 1990) designed for this project.

The algorithm relies on neighbouring beliefs exchanging messages constructed from their own direct evidence (if any), messages received from other neighbours

[6] *Beta distributions* are continuous probability distributions with probability density functions defined on the finite interval $[0, 1]$, commonly used in Bayesian statistical inference (Wikipedia, 2006).

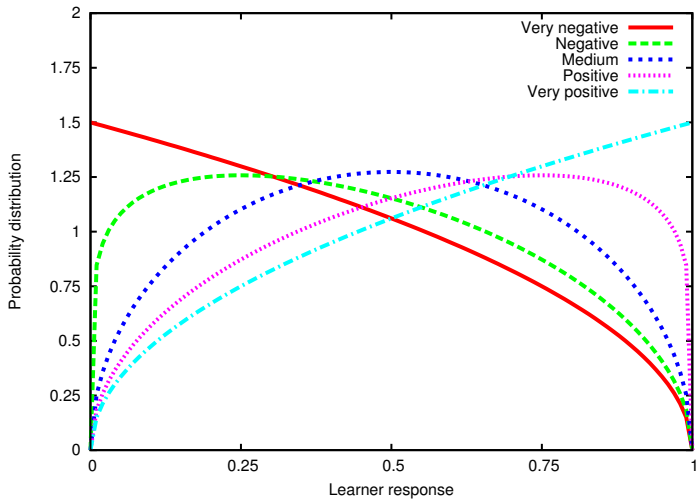


Figure 4.7: Beta distributions used by LM for each value that can be reported by learners using the Self-Report Tool. The intervals $[0, 0.25]$, $[0.25, 0.50]$, $[0.50, 0.75]$ an $[0.75, 1.0]$ correspond to the levels I, II, III and IV, respectively.

and (conditional) mass functions in the association between neighbours. In every iteration, all beliefs that have received updated messages (with adjusted evidence) from its neighbours re-calculate their own and check whether these have changed significantly (given a predefined threshold and a method for comparing mass functions). If that is the case, and their messages are not full with ignorance (not beyond another predefined threshold) then they send them to their neighbours. The iterative process ends when no more messages have been exchanged or when a predefined maximum number of iterations have been reached.

Threshold are included in the algorithm to cope with the complexity of the calculations, particularly for the case of larger maps for the dimensions in the learner modelling framework. All thresholds are configurable using JAVA properties.

4.3.4 Belief updating

At the end of the propagation of evidence, many beliefs in a learner model have new evidences ascribed to them. These evidences are shaped as pairs of the form $(eventId, m_i)$, where $eventId$ is the identifier given to the event that produced the evidence and m_i is a mass function corresponding either to the (direct) interpretation of the event or to a message received from the neighbouring belief. The next step is to combine all new evidence into a single mass function m_e, and this is done using TBM combination operator $\oplus$,

$$m_e = m_1 \oplus m_2 \oplus \cdots \oplus m_n \tag{4.8}$$

defined using the *combination rule*

$$m_i \oplus m_j(s) = \sum_{a \cap b = s} m_i(a) \times m_j(b). \tag{4.9}$$

This combination rule produces an operator that is associative and commutative, allowing its iterative application as in equation (4.8). It is the combination rule used all over LM every time mass functions (representing either beliefs or evidences) need to be combined. It is the same rule as in Dempster-Shafer Theory (DST), except for the fact that $m(\emptyset)$ is never forced to be zero. The mass assigned to the empty set is interpreted in xLM as the amount of conflict in the evidence.

The final step of the process of updating beliefs consist in combining the mass function that represents the belief m_b with the mass function representing the new evidence accumulated m_e. This is done by discounting m_b first—making older evidence weaker—and then combining it with m_e using the $\oplus$ operator,

$$m_b \leftarrow d \times m_b \oplus m_e, \tag{4.10}$$

where d is is a value in between zero and one.

4.4 Learner History

The learner history needs to store received events, which are JAVA objects, in a database. The problem of storing such objects in a relational database has already been subject to much attention, and several solutions that tackle this problem are freely available—for more information about the object-relational paradigm mismatch, we refer to an overview article by Ambler (2006). The problem is harder than it might seem at first glance. What we were looking for was a configurable persistence layer, i.e. a framework that encapsulates the connection to the persistence engine, which would take care of storing and retrieving events in a database.

The learner history implementation is based on HIBERNATE, which is a framework for object/relational persistence. HIBERNATE's query language, HQL, is independent of the database in use. It is not hard to learn and very similar to SQL, but avoids dealing with SQL dialect issues. It can be connected to almost any database and features its own query language. HIBERNATE is completely transparent regarding the database it is connected to.

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. Figure 2 shows a sample mapping of an event. HIBERNATE manages the persistent objects by using a session mechanism. Objects can be persisted within such a session by simply using the session's `save` method. Of course, hibernate has much more features to manipulate the persistence life cycle of objects, but because of the relatively simple structure of the events, there was only one important decision to take in this respect, namely what the database schema should look like. This is an important issue in object/relation mapping. HIBERNATE offers three strategies to deal with subclasses like our events:

Listing 2: A sample HIBERNATE XML mapping.

```
<joined−subclass
  name="org.activemath.events.types.PagePresentedEvent"
  table="eventPagePresented">
    <key column="eventId"/>
    <property name="bookId" column="book"/>
    <property name="page" column="page"/>
</joined−subclass>
```

- Table per class, which means that a class is completely mapped to its own table, even if it shares some fields with the base class..
- Table per class hierarchy, which means that a "big" table is created that contains all fields of all classes in the hierarchy.
- Table per subclass, which means that a table is created for the base class and for each subclass.

Each of the three approaches has advantages and drawbacks. We decided to use the table per subclass strategy because table per class hierarchy generates a single table that is hard to manage, and table per class generates too much redundant columns (one for each attribute of the base class). The schema created by the table per subclass approach is clean and quite flexible. To add a new event, only a table for the new subclass needs to be created.

An important feature of HIBERNATE is the retrieval of stored objects is by using a so-called *criteria*-based approach. The idea behind this approach is to set criteria, i.e. constraints on attribute values, in order to retrieve matching objects. Thus, we do not always need to design an SQL or HQL query to fetch events from the database.

LH query interface uses the query by example approach, it has a method that takes a partial event instantiation as an argument. This partial instantiation is then translated into a HIBERNATE criteria-based query, which can readily be used to retrieve the desired events. HIBERNATE provides the tools to easily perform the query translation. This method has been built for the typical use case of a component that needs to know whether the user already performed an action (e.g. managed to solve some specific exercise).

One more level of detail to query the LH is provided by the object `HistoryQuery`. This object can be used to explicitly set the constraints on event attributes, similar to the partial instantiation of an event. In addition to the attributes available for partial instantiation, the `HistoryQuery` includes special constraints on numerical attributes. This is useful to express criteria such as "events the user has generated in the last 5 minutes" or "exercises the user solved at least 50 percent". `HistoryQuery` objects are internally translated into a HIBERNATE criteria query.

The implementation of the Learner History was made corresponding to the specifications set up in deliverable D10 (Andrès et al., 2005), Student Model Specification. For in-depth information about the learner history, please consult this document.

Chapter 5

Outstanding issues

So far this report has described the diagnostic functionality of the Extended Learner Model, the new learner modelling subsystem for LEACTIVEMATH. xLM tries to capture the multi-dimensional nature of learners, their ongoing states and dispositions using a combination of object oriented programming, concept maps, production rules and approximate reasoning, the latter in the form of belief networks—both probabilistic and based on belief functions, defined on top of the concept maps that define the modelled learner dimension. The diagnostic and modelling processes inside xLM have been sketched with details and examples.

This final chapter aims to discuss outstanding issues concerning obstacles that have interfered with developing the diagnostic functionalities of xLM, as well as limitations of the current prototype worthwhile further research and development.

Knowledge vs content From the beginning of the project there have been discrepancies regarding the nature of the material developed for the project in OMDoc. From one viewpoint, it can be seen close to mathematical knowledge, specially given OMDoc semantic nature. From another viewpoint, the semantic nature of OMDoc is enhanced or diminished by the nature of the documents it encodes and the processing capabilities of the interpreters.

Formal mathematical documents encoded in OMDoc are written with consistency and completeness in mind. Their purpose is to represent knowledge that can be verified, proved and otherwise interpreted and used by computers. On the other hand, educational documents are written pedagogically, their purpose being to provoke learning experiences, which themselves are not usually represented explicitly in the document. Educational documents and collections of them can be rather inconsistent, repetitive and incomplete, even on purpose if that is believed to improve their pedagogical effect.

The issue gets acute when it comes to decide the shape for the subject domain map in xLM. One possibility is to use the content available as a map, with content items (e.g. OMDoc concepts and symbols) being subjects of beliefs. On one hand, the approach is quick and simple, and it is the one used by ACTIVEMATH old learner model. Every time a new piece of content is created, a new subject domain topic for xLM to have beliefs on is created as well. Every author can

define topics for xLM to model learners on. On the other hand, it is an approach prone to inconsistencies, repetitions and incompleteness, very much as content is. Another possibility is to develop an *explicit* ontological or conceptual map of the subject domain, as a more stable framework for xLM to ground beliefs on. Given the lack of a domain expert embedded in LEACTIVEMATH, able to interpret content and extract knowledge from it, a map of the domain delivers part of the hidden, implicit content semantic. A map of the domain can help authors to better describe their content. They can make explicit references to it.

A separate concept map for the subject domain is the implementation of choice for xLM. It provides a solid ground for learner modelling which is less sensitive to changes in content. However, it has had almost no support elsewhere in the project. Consequently, the subject domain map is currently implemented as part of xLM and covers only the minimum of Differential Calculus necessary for LEACTIVEMATH evaluation.

Misconceptions Until very recently, misconceptions identified and catalogued by authors had been introduced in content and content metadata only as comments, with no formal representation nor mechanisms for their retrieval. Some additional work has been done following the last project meeting in Málaga, Spain, yet misconceptions representation in LEACTIVEMATH is still very primitive, a list of `OMDoc` symbols with no references to other content items.

This condition have severely affected xLM processing of misconceptions, here called *conceptual and procedural errors* (CAPE). The framework is set for their processing, yet there has been no material to evaluate it properly

Competency, level and difficulty In PISA, competency levels are holistic entities, competency clusters that reflect 'how mathematical processes are typically employed when solving problems' (OECD, 2003, p. 31). Their operationalisation in xLM reverses the relationship between competency and competency level, the latter becoming the scale to measure progress in the former. The idea is that, when considered in conjunction, the collection of competencies and their grades will produce a holistic competency level. However, this is an idea still to be explored and evaluated.

In the same vein, competency level and difficulty are interpreted by xLM as two granularities in the same scale. Very much like metres and centimetres. This may not do justice to the nature of competency levels, which seems to represent rather big qualitative stages but, again, is a way of operationalising them for learner modelling.

Metadata usage A core but minimum subset of the available metadata for content is actually taken into account while interpreting events, and could be expanded for better. Nonetheless, most metadata of the current LEACTIVEMATH

content has been produced based in the subjective appreciation of their authors, rather than on empirical evaluation of content, and it is suspected that there is a strong correlation between their different values. In such a case, taken into account more elements of metadata can be misleading.

Interpretation of events The probability assignment functions and probability distributions used by LM (figures 4.6 and 4.7, respectively) are aimed to take into account the author's subjectivity in metadata and the uncertainty of interpreting learner behaviour. In the current implementation, the same set of functions are used for interpreting all events, independently of the source, which means they are all given the same credibility. This is the case, for example, for the learner using the Self-Report Tool and the Situational Model. Since both produce diagnostic events, the same Beta distributions are used to interpret their input by LM. OLMMetacog events produced by the Open Learner Model and ExerciseFinished events are interpreted using the same probability assignment functions. Each case needs to be refined as experience of xLM use accumulates.

Propagation algorithm A careful analysis of the propagation algorithm is necessary to determine suitable adjustments and the requirements on the maps that would guarantee it to converge to a good approximation of the correct marginals. On the same line, there are a few parameters that can be fine tuned to optimise xLM performance in terms of accuracy, reliability and efficiency. Of particular interest is the issue of performance with larger maps for the different layers of the learner modelling framework.

Bibliography

ADL (2004). *Sharable Content Reference Model (SCORM) 2004: Overview*. Advanced Distributed Learning, 2 edn.

Ambler, S. W. (2006). The object-relational impedance mismatch. URL `http://www.agiledata.org/essays/impedanceMismatch.html`.

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

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

Brown, A. (1987). Metacognition, executive control, self-regulation, and other more mysterious mechanisms. In F. E. Weinert and R. H. Kluwe, eds., *Metacognition, Motivation, and Understanding*, pp. 65–116. Lawrence Erlbaum Associates.

Brown, P. and S. C. Levinson (1987). *Politeness: Some Universals in Language Usage*. No. 4 in Studies in Interactional Sociolinguistics. Cambridge University Press.

Conati, C., A. Gertner and K. VanLehn (2002). Using bayesian networks to manage uncertainty in student modeling. *User Modeling and User Adapted Interaction*, 12(4) 371–417.

Corbett, A. T. and J. R. Anderson (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User Adapted Interaction*, 4(4) 253–278.

DSL (1999). *SMILE: Structural Modeling, Inference, and Learning Engine. Apllication Programmer's Manual*. University of Pittsburgh.

Fung, R. and B. Del Favero (1994). Backward simulation in Bayesian Networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pp. 227–234.

Godauze, G. (2005). Structures and metadata model. Deliverable D6, LeActiveMath Consortium.

Guo, H. and W. Hsu (2003). A survey of algorithms for real-time Bayesian Network inference. URL `citeseer.nj.nec.com/552206.html`.

Heckermann, D. (1996). A tutorial on learning with Bayesian Networks. Tech. rep., Microsoft Research.

Henrion, M. (1988). Propagating uncertainty in Bayesian Networks by probabilistic logic sampling. *Uncertainty in Artificial Intelligence*, (2) 149–163.

IEEE (2002). Draft standard for learning object metadata. Tech. Rep. IEEE 1484.12.1-2002, IEEE.

Jameson, A. (1996). Numerical uncertainty management in user and student modeling: An overview of systems and issues. *User Modeling and User Adapted Interaction*, 5(3–4) 193–251.

Jensen, F. V. (1996). *An Introduction to Bayesian Networks.* Springer Verlag, New York.

Lauritzen, S. L. and D. J. Spiegelhalter (1988). Local computations with probabilities on graphical structures and their applications to expert systems. *Journal of the Royal Statistical Society*, 50(2).

LeActiveMath (2004). LeActiveMath contract – annex i: Description of work.

Libbretch, 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.

Ortony, A., G. Clore and A. Collins (1988). *The Cognitive Structure of Emotions*. Cambridge University Press, Cambridge.

Polson, M. C. and J. J. Richardson, eds. (1988). *Foundations of Intelligent Tutoring Systems*. Lawrence Erlbaum Associates, New Jersey.

Sentz, K. and S. Ferson (2002). Combination of evidence in dempster-shafer theory. Sandia Report SAND2002-0835, Sandia National Laboratories.

Shachter, R. D. and M. A. Peot (1990). Simulation approaches to general probabilistic inference on belief networks. *Uncertainty in Artificial Intelligence*, (5) 221–231.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

Shenoy, P. P. and G. Shafer (1990). Axioms for probability in belief-function propagation. In R. D. Shachter, T. S. Levitt, L. N. Kanal and J. F. Lemmer, eds., *Proceedings of the Fourth Anual Conference on Uncertainty in Artificial Intelligence*, pp. 169–198. North-Holland.

Smets, P. and R. Kennes (1994). The transferable belief model. *Artificial Intelligence*, 66(2) 191–234.

Stillings, N. A., S. E. Weisler, C. H. Chase, M. H. Feinstein, J. L. Garfield and E. L. Rissland (1995). *Cognitive Science: An Introduction*. MIT Press, 2 edn.

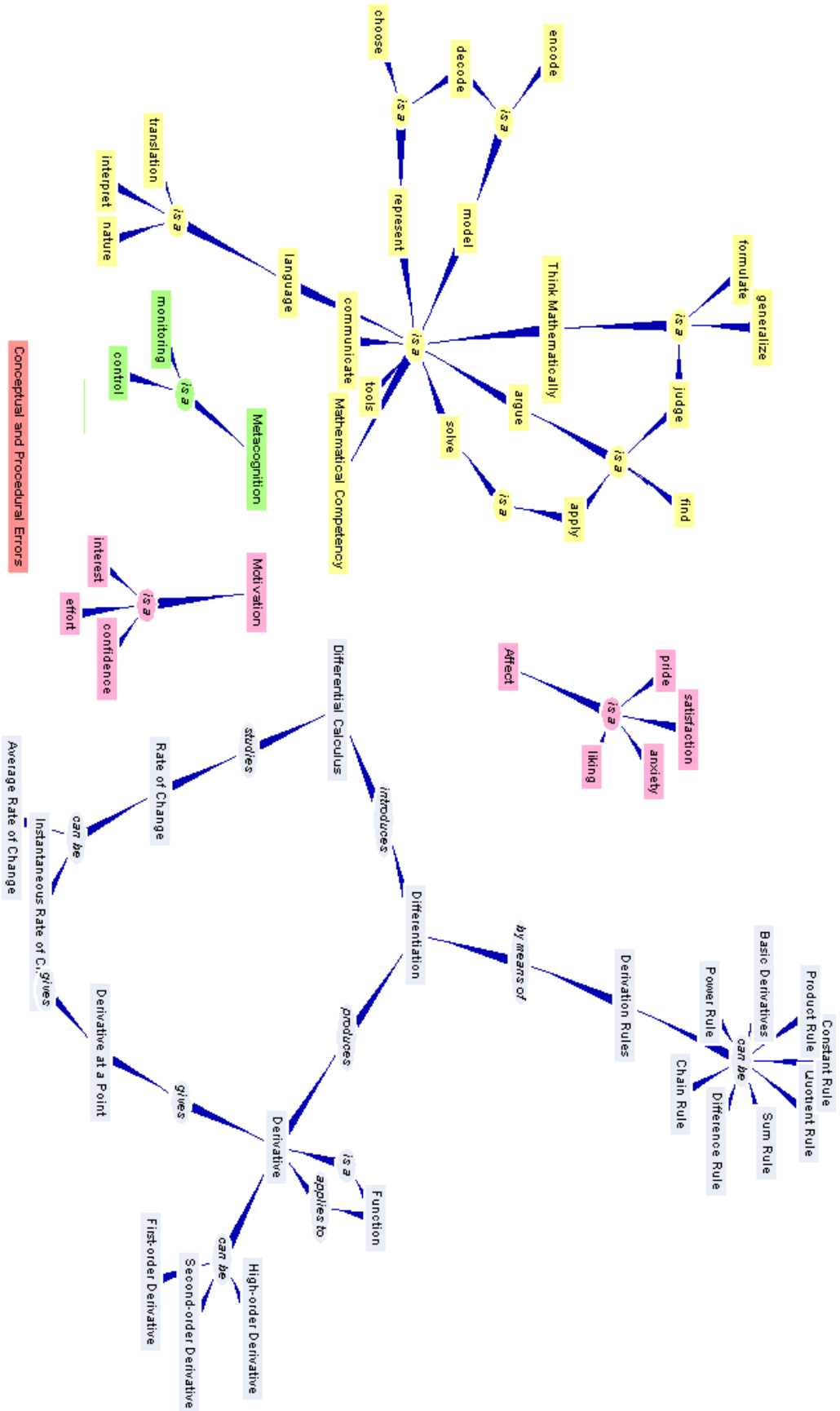
Wikipedia (2006). Beta distribution — wikipedia, the free encyclopedia. URL `http://en.wikipedia.org/w/index.php?title=Beta_distribution&oldid=35219983`. Accessed on 26 January 2006.

Zapata-Rivera, J.-D. and J. E. Greer (2000). Inspecting and visualizing distributed bayesian student models. In G. Gauthier, C. Frasson and K. VanLehn, eds., *Intelligent Tutoring Systems: Fifth International Conference, ITS'2000*, 1839, pp. 544–553. Springer-Verlag.

Appendices

Appendix A

Learner Model Maps


Mathematical Competency
is a
communicate
represent
model
tools
language
translation
interpret
nature
choose
decode
encode
Think Mathematically
formulate
generalize
judge
argue
solve
find
apply
Metacognition
monitoring
control
Conceptual and Procedural Errors
Motivation
interest
effort
confidence
Affect
pride
satisfaction
anxiety
liking
Differential Calculus
studies
introduces
Rate of Change
can be
Average Rate of Change
Instantaneous Rate of C.
gives
Differentiation
by means of
Derivation Rules
Power Rule
Chain Rule
Basic Derivatives
Constant Rule
Product Rule
Quotient Rule
Sum Rule
Difference Rule
produces
Derivative at a Point
gives
Derivative
applies to
Function
can be
First-order Derivative
Second-order Derivative
High-order Derivative