

Continuous Learner Modeling in iClass

Martin Muehlenbrock, Stefan Winterstein, Eric Andres, Andreas Meier
German Research Center for Artificial Intelligence (DFKI)
Email: Martin.Muehlenbrock@dfki.de

Abstract: The recently launched European Integrated Project iClass is developing a learning system that takes into account the multi-cultural and multilingual characteristics of its member countries in pre-university education. For adapting the presentation of learning objects to the learners' needs, the iClass system will keep track of the students' domain knowledge. The learner modeling system consists of three components, i.e. the Tracker, the Profiler, and the Monitor, with the Tracker receiving, storing, and distributing all learning-related events and the Profiler identifying learners' preferences and cultural specifics. Finally the Monitor creates and updates the model of the learner's knowledge based on assessment as well as continuous knowledge level monitoring. The monitor incorporates an elaborate student model that represents mastery in terms of knowledge, comprehension, and application for each concept, which are updated from reading and problem solving activity.

Introduction

United under a consortium, 22 partners from 11 different European countries are working to develop the next generation of learning systems at a European level under the flag of iClass. These partners include leading IT companies such as Siemens Business Services, Microsoft, Sun, and Intel, as well as Europe's leading universities, research organizations, and school networks. Initiated in January 2004, iClass will involve the development of a system and standards that take into account demographic, multicultural, and multilingual characteristics of the member countries in pre-university education. The system will enable personalization of learning according to different cultural characteristics, existing knowledge level of learners, and learning styles.

The results of iClass will include not only an intelligent learning system and environment, but also standards for e-education and e-content development, ensuring efficient use of technology in education. The efforts will, by envisioning the educational process of the future, involve the following: (1) extensive research to define and design the next generation products and services, (2) creation and testing of prototypes in school settings, (3) and the measurement of the impact to ensure effective utilization of technology in pre-university education. Current efforts in iClass focus on the research requirements with a vision that the results of the project will start to be utilized from 2010 on.

Learner Modeling

In order to provide learner-oriented individualization, the iClass system will need to keep track of the student's domain knowledge across sessions, i.e. the students' data has to be stored and has to be made accessible to the system for a longer period of time. An important question is how the students' knowledge about the domain can best be represented and updated. There are a number of approaches to tackle this problem.

The classical approach is to model the knowledge of an expert about the domain, and use a subset of this as the student model, the so-called overlay model. An enhancement of this, i.e. a differential model, also takes into account the knowledge the student is expected to have at a given time during the course. A significant weakness of the overlay-based approach is that it does not support the representation of the student's incorrect beliefs. The perturbation models take incorrect beliefs possibly occurring in the domain in consideration, and treat the student model as a subset of the expert knowledge and the incorrect beliefs (commonly referred to as mal-knowledge). A major problem with this approach is to identify to make the mal-knowledge available to the student model. There are machine learning approaches that identify incorrect knowledge quite well. The category of perturbation models can

again be split into subcategories depending on how they handle the mal-knowledge. Furthermore, there are the stereotype-based models. These are based on a fixed number of stereotypes consisting of a set of assumptions about the students. Each student is associated to one of those stereotypes. This approach is highly inflexible, but it can be useful for domains where knowledge can not be split up into atoms.

An intelligent tutoring system (or an intelligent learning environment) needs to maintain a model of the student during problem solving. This model contains detailed information about the beliefs of the student while she's working on a problem. Some of the beliefs may become irrelevant after the student finishes her work, so the short-term model just transfers relevant information into the more abstract long-term model. We can distinguish between the model-tracing approach and the constraint-based approach, which we present in more detail in the following. A striking difference between the two approaches is that model-tracing models declarative knowledge as well as procedural knowledge, whereas constraint-based models do not include procedural knowledge.

With the rise of open internet-based educational systems, there came also the wish to exchange student model data between these systems. Indeed, the internet provides the possibility to transfer data between such systems. However, each of those adaptive hypermedia systems usually has its own special representation of the student model content, and this prevents the reuse of information. Thus, there have been attempts to standardize the learner model contents. Two major standards have emerged from these efforts, the IEEE Personal and Private Information (PAPI), and the IMS Learner Information Package (LIP, <http://www.imsglobal.org/>). These standards have been written from two different points of view. In the PAPI standard, performance information is considered as very important, but there are also other information contained. The LIP standard is derived from the classical CV notion.

In LIP, we have the several slots that represent relevant information about the learner. The slot identification contains biographic and demographic data. Competency is used to mention the learner's skills, knowledge and abilities. The Goal slot contains the learner's goals and objectives. Accessibility defines the general accessibility to the learner information. The qcl slot specifies the qualifications, certifications and licenses. Transcript is a summary of the academic achievement. The Activity category is used to represent any learning activity in any state of completion. Affiliation contains the membership of professional organizations. The interest slot expresses the user's miscellany interests. Securitykey contains passwords and security keys assigned to the user. The PAPI standard represents the learner information using the categories depicted above. The Learner Contact contains the learner's contact information. Learner Relations represents the learner's relationships to other relevant persons. The Learner security slot is about the learner's credentials. Learner Preference describes preferences relevant to the human-computer interaction process. Learner Performance contains information about the learner's history, objectives and current work. The Learner Portfolio is a collection of representative achievements.

Learner Model Architecture

Events are things of interest that happen at a certain point in time (and space). In iClass, we follow the established publisher/listener paradigm: the entity producing an event can publish it, and all entities who have registered as listeners will receive the event. For the learner modeling subsystem, events are the major source of information coming in for the learner modeling components. Events allow for an asynchronous coupling with the rest for the iClass system. The learner modeling components only handle events related to a user or learner. Other kinds of events that exist in the iClass system such as the availability of new content are not considered. Therefore, the terms "event" and "learner event" are used interchangeably in the learner modeling subsystem. All learner events have the following common properties:

- a **learner** (what user is this related to),
- a **timestamp** (when did it happen),
- a **publisher** (who produced the event), and
- an event might have a **context** (where did they happen, e.g. a session).

Additionally, events also carry information specific to the type of the event. For example, an event indicating that a learner has seen an item of a learning object (LO) will contain references to that learning object item.

The learner modeling system consists of three components, i.e. the Tracker, the Profiler, and the Monitor. The Tracker module will be the single point of entry for events entering the learner modeling subsystem. All events tied to a learner are captured in the history log of that learner. As a special case, learner interaction events have to be propagated to the Monitor, since they form the basis for the knowledge model. The event propagation is illustrated in the sequence diagram Figure 1.

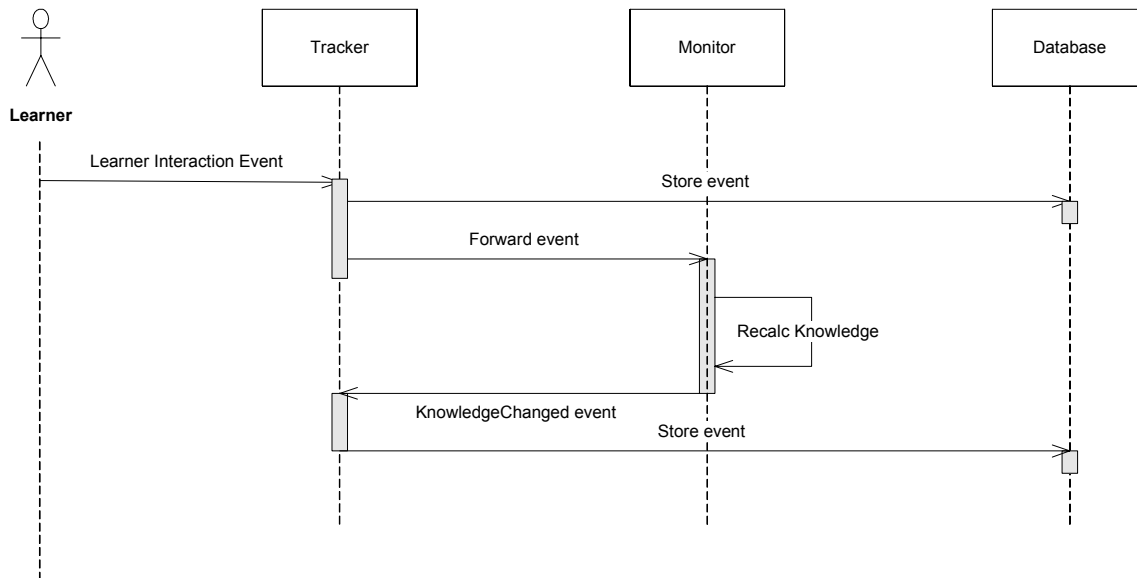


Figure 1. Propagation of events

Upon receipt of a learner interaction event, the Tracker stores the event in the learner's history like any other event. The event is then forwarded to the Monitor. In the Monitor, the receipt of a learner interaction event triggers an update of the learner's knowledge model. This recalculation might result in changes to the learner's knowledge level for a concept. If this is the case, the Monitor sends a knowledge event to the Profiler. In case the knowledge change affects several concepts, a separate event is sent for each change. Content Modification events are forwarded in the same manner and trigger the Monitor to rebuild its internal knowledge structures tied to the content.

The Monitor maintains a model of the assumed learner's knowledge. It implements two approaches to knowledge modeling which complement each other, i.e.

- a preliminary assessment of concepts based on Knowledge Space theory based as described in (Albert and Hockemeyer, 1997, Hockemeyer and Albert, 1999), which will be repeated on a regular basis to update the learner model, and
- a continuous knowledge level monitoring based on concepts, taking into account diverse inputs such as exercise results, reading time, and possibly confidence degrees.

The latter will be described in more detail in the following section. Finally, the Profiler is responsible for the management of the learners' profiles, i.e. by using questionnaires it collects data about the learning style and preferences of a learner, and it evaluates this data and generates a learning style profile for each learner. The Profiler also inquires about demographic data such as name, surname, birthday, etc. of a learner and some other user preferences such as cultural preferences or special needs. The Profiler module stores the learner profiles in a repository in an exchangeable and interoperable form that also complies with existing e-Learning standards.

Continuous Knowledge Level Monitoring

The Monitor also maintains a knowledge model of the learner based on a knowledge level for the relevant concepts. It is assumed that most valid assumptions on a learner's knowledge can be obtained by monitoring the learner's

performance in interactive exercises. They allow for a concrete assessment of learner knowledge. Another source of information are reading actions, i.e. how long the learner has been reading a text. Reading actions are less reliable than exercise actions and only allow for a rough estimation of a learner's knowledge. While the learner is interacting with the learning objects, the Monitor continuously updates the knowledge model of the learner. The tracker notifies the Monitor of a learner interaction event with an LO (e.g. "the learner has seen an LO for concept C for time t", "has solved an exercise for concept C to a percentage of x"). The Monitor recalculates the knowledge model of the learner according to its updating policy. If the knowledge model changes, the Profiler is notified of the change.

The Monitor incorporates a very elaborate student model that stores explicit information about what learning materials the learner has studied and his mastery of these learning materials. The mastery is represented as a triple of knowledge, comprehension, and application, values that represent a subset of Bloom's taxonomy of educational objectives (Bloom, 1956). The actual the representation of the learner model is derived from Shute's "Taxonomy of outcome types" (Shute, 1995). The knowledge level of a concept c , i.e. $KL(c)$, is a triple (K, C, A) of three distinct values, i.e.

- the **knowledge** of a concept (K),
- the **comprehension** of a concept (C), and
- the **application** of a concept (A).

Each of the three values stands for the probability that a learner masters knowledge, comprehension, and application of that concept, respectively. The values are called mastery values. Each of the three mastery values is in the range of $[0...100]$. The lowest KL for a concept is $(0, 0, 0)$, the highest KL is $(100, 100, 100)$. The Monitor maintains a knowledge level for each concept that is present in the content. The knowledge level is persistent (i.e. stored in a database). Whenever new content is added to the system, the Monitor is notified about this. The initial knowledge level for a concept appearing newly in the content is $(0, 0, 0)$. Whenever a learner sees an LO item related to a concept C or solves an interactive item (e.g. an exercise) correctly, the KL s for the related concepts are increased. The KL s are decreased when an interactive LO item is not solved correctly. For each evaluation of a learner interaction, the knowledge level is updated according to this basic formula:

$$KL_{i+1}(c) = KL_i(c) + \Delta KL(event)$$

The calculation of $\Delta KL(event)$ is described in the following paragraphs. When adding ΔKL to KL_i the following rules apply:

- The three mastery values of ΔKL are added separately to their corresponding values in KL_i .
- When adding a mastery value, the result is cut off at 100 (overflow). When subtracting a value, the result is cut off at 0. (underflow)
- When ΔKL contains a positive value for a mastery value of KL_i that has already reached 100, no value of ΔKL is added, and the knowledge level is not changed.

Reading

The event reporting a reading action is the `loItemSeen` event. It contains the id of the LO item in question, the concepts it relates to, and the time how long the learner has read this item. Using this data, the following expression is calculated for ΔKL :

$$\Delta KL(loItemSeenEvent) = \Delta KL(LO.class) * speedWeight * difficultyWeight * iterationWeight$$

Each class of LOs targets a different change of the mastery values. The values for $\Delta KL(LO.class)$ represent the degree in which LOs of certain classes contribute to the knowledge level of a certain concept in terms of knowledge, comprehension, and application. That is, reading an LO item of class "comprehension" (such as an explaining text) mainly targets the comprehension mastery value, and less the values for knowledge and application.

The reading speed is classified according to the optimal reading time estimate as outlined in [Conati99]. The speed of reading an LO is defined as

$$readingSpeed = \# \text{ of words in LO} / (\text{duration} / 60) \quad (\text{i.e., words per minute})$$

Here, *duration* is the value from the *loItemSeen* event indicating how long (in seconds) the learner has seen or read the LO. The reading speed (words per minute) is classified into 4 classes and translates into a speed factor as shown in table 1.

Reading Speed	Classification	Speed Weight
low threshold $\leq readingSpeed < \text{high threshold}$	OK	$speedWeight = 1$
high threshold $\leq readingSpeed < \text{upper bound}$	SHORT	$speedWeight < 1$
lower bound $\leq readingSpeed < \text{low threshold}$	LONG	$speedWeight > 1$
else	INVALID	$speedWeight = 0$

Table 2. Classification of reading speed

The assumption is that reading a text thoroughly yields a better increase in *KL* than just skimming over it. An unreasonably slow or quick reading is not taken into account (no reading actually took place, or duration might be wrongly measured).

The difficulty of an LO item is classified into 3 classes that take into account the available content metadata. Important parameters are metadata on difficulty, density and abstractness. Each difficulty class is mapped to a difficulty factor *difficultyWeight*. For reading actions, the following assumptions are made: easy (*difficultyWeight* > 1), medium (*difficultyWeight* = 1), and hard (*difficultyWeight* < 1). The idea is that reading an easy text yields in a higher increase in *KL* than reading a text that is hard to understand.

How many times a learner has already seen a text before is also relevant to the calculation of the new *KL*. This information can be obtained via the Tracker API. The assumption is that the change in *KL* should decrease with the number of times the LO item has been seen. For an LO item that has never been read the *iterationWeight* initially is 1 and decreases successively with the number of reading times. A finer grained estimate of a learner's reading actions in terms of subitems of a page can be achieved by using eye trackers, with the poor man's eye tracker DFKeye being an interesting option that can do without additional hardware (Ullrich, Wallach, and Melis, 2003)

Solving an exercise

The calculation of ΔKL when the learner has solved an exercise/interactive LO is in principle similar to the one when reading as described above, i.e.,

$$\Delta KL(\text{loItemSolvedEvent}) = \Delta KL(\text{loItemSolved}) * successWeight * difficultyWeight * iterationWeight$$

Solving an interactive LO should give a strong boost to the application mastery value. In contrast to reading actions, this does not depend on the LO class, which must be of type "Application". In contrast to just reading about a concept, solving an exercise wrongly can result in a decrease of mastery values. It is based on the success degree indicated by the *loItemSolved* event (which is in the range of [0...100]). The success factor is calculated as follows:

$$successWeight = (\text{loItemSolvedEvent.success} - 50) * 2 / 100$$

Therefore, the value of *successWeight* is in the range [-1.0; 1.0]. The difficulty of an interactive LO is classified into three classes (details to be decided based on LO metadata). Each difficulty class is mapped to a difficulty factor. For solving actions, we suggest the following values: easy (*difficultyWeight* < 1), medium (*difficultyWeight* = 1), and hard (*difficultyWeight* > 1). The underlying assumption is that solving a difficult exercise correctly indicates a high knowledge level. How many times a learner has already tried an exercise before is again relevant to the calculation of the new *KL*, with the idea that the change in *KL* should sharply decrease with the number of times the LO has been tried.

Further Work

This paper describes work in progress. The learner modeling system has been specified and is currently implemented. First system testing has started at the beginning of 2005, and evaluation involving different European schools is planned for mid 2005.

Acknowledgement

The work presented in this paper is partially supported by European Community under the Information Society Technologies (IST) program of the 6th FP for RTD - project iClass contract IST-507922.

References

Albert, D., Hockemeyer, C., (1997). Dynamic and adaptive hypertext tutoring systems based on knowledge space theory. In Benedict du Boulay and Riichiro Mizoguchi, editors, *Artificial Intelligence in Education: Knowledge and Media in Learning Systems*, pp. 553-555, Amsterdam. IOS Press.

Bloom, B.S., editor (1956). *Taxonomy of educational objectives: The classification of educational goals: Handbook I, cognitive domain*. Longmans, Green, New York, Toronto

Conati, C., VanLehn, K. (1999). Teaching meta-cognitive skills: Implementation and evaluation of a tutoring system to guide self-explanation while learning from examples. In S.P. Lajoie and M. Vivet, editors, *Artificial Intelligence in Education*, pages 297–304. IOS Press.

Hockemeyer, C., and Dietrich Albert, D., (1999). The adaptive tutoring system RATH. In Michael E. Auer and Ursula Ressler, editors, *Workshop on Interactive Computer aided Learning: Tools and Applications at ICL99*, Carinthia Tech Institute, Villach, Austria.

Shute, V.J. (1995). Smart: Student Modeling Approach for Responsive Tutoring. *User Modeling and User-Adapted Interaction*, 5(1): 1-44.

Ullrich, C., Wallach, D., Melis, E. (2003). What is Poor Man's Eye Tracking Good For? *17th Annual Human-Computer Interaction Conference HCI-2003*, Bath, UK.

Ullrich, C., and A. Chen, A. (2004). An Easily Implementable Method to Support Goal-Directed Learning, *Proceedings of the World Conference on Educational Multimedia, Hypermedia and Telecommunications ED-MEDIA 2004*, Lugano, Switzerland.