

A RAPID EDUCATIONAL SOFTWARE DEVELOPMENT

Aury de Sá Leite¹, Galeno José de Sena¹ and Maria de Fátima C.L. Santos¹

Abstract $\frac{3}{4}$ This paper shows the researches and the preliminary results in a domain-based software architecture development: an educational software family. The analyzed domain is the pedagogical field, intending the cooperative learning involving teachers, learners and computers. The adopted concepts and the methodologies for the consequent software development, according to the pedagogical domain model here proposed, are: Rapid Application Development (RAD Methodology); software development-for-reuse; domain-based software architecture; Conceptual Adherence and software family development.

Index Terms $\frac{3}{4}$ Rapid Application Development; domain-based software architecture; Conceptual Adherence; educational software family development.

INTRODUCTION

Nowadays teachers can develop their own scientific, academic or educative softwares. The Rapid Application Development (RAD) systems enable programmers to quickly build software applications. RAD systems provide tools to easily build sophisticated graphical user interfaces and provide facilities to the background programming. Borland Delphi, Borland C++ Builder and Visual Basic are, for instance, any of the popular RAD systems for Windows development. These RAD Systems uses as background Object Oriented (OO) languages, respectively: Pascal, C++, and Basic. In fact, these environments are *Windows-Event Oriented Languages*. Their main characteristic is: You not may use or not know the OO methodology to obtain useful software systems with satisfactory performance.

RAPID SYSTEMS DEVELOPMENT

The main target of RAD systems is to develop software products fasten and with high quality. RAD systems have the following characteristics:

- Incorporates good user interface that simplifies programming tasks, thereby increases the usability.
- Reduces training costs and increase productivity through simple and easy to use interfaces.
- Allows testing many of the users/customers requirements through prototyping, already in the initial phase of the project.
- Allows to assemble user interfaces or system managers panels using Windows objects, directly on the screen of RAD systems, using “drag-and-paste”.

- Enables the repetitive develop-and-test of all phases of the software development for any project associated person.
- Promotes and stimulates the immediate individual develop-and-test.
- Diminish maintenance costs through the modularization imposed by Event Oriented Languages.
- Allows, through continued prototyping, the creation of a sequence of prototypes, which describes the evolution of ideas and can show the new features by comparisons between prototypes.
- Encourages and facilitates both the constant peer reviews and the repeatedly (again and again) users reviews during all the software development phases.
- Demands less formality in reviews and other team communication, diminishing software development time and costs.
- Rejects a rigidly paced schedule that defers design improvements to the next product version facilitating the software evolution.

RAD Systems and Associated Tools

Some companies develop and offer products for RAD software development, which inherently fosters software reuse. These products can include gathering tools, prototyping tools, CASE (Computer Aided Software Engineering) tools, GroupWare (software applications that facilitate shared work over long distances on documents and information) for communication among development members, and testing tools. In general, products for RAD software development usually embraces object oriented programming methodology. The most popular object-oriented programming languages, Pascal, Basic, C++ and Java, are offered in efficient visual programming packages. These packages can increase software productivity because they are easy to learn and easy to use; compiles quickly and creates fast executable code.

SOFTWARE REUSABILITY

The software reusability techniques have growing in importance in the last decade. The software *development-with-reuse* and the *development-for-reuse* are pointed nowadays as keys to reduce time and costs of software system construction and maintenance. While the *development-with-reuse* is not a well-indicated technique in many cases, the *development-for-reuse* is a good option

¹ UNESP – Universidade Estadual Paulista “Júlio de Mesquita Filho” – Campus de Guaratinguetá, São Paulo – BRAZIL {aury;gsena;flacaz}@feg.unesp.br

when we intend future expansions for the system capabilities through new sub-systems aggregation.

Reusability involves, by-extension, but generally, by necessity, the Domain Analysis methodologies and the Software Architecture theories. We will examine these concepts in the next sections.

ABOUT DOMAIN AND DOMAIN ANALYSIS

We can interpret the word domain as a territory over which rule or control is exercised, a sphere of activity, concern, or function; a field of knowledge or activity characterized by a collection of concepts that involves specific words, that is, an experts terminology.

On the other hand, when we talk about Software Engineering products, another interpretation, for that word, sometimes referred as "software domain" is: a set of technological solutions and specific designs selected to meet the specific software requirements based on ideas, solicitations and suggestions of the users, experts, customers or a particular group of customers.

Analyzing these two situations we can conclude that:

- *In the first case a domain refers to a specific body of knowledge and, in the second case, domain comprises a field of knowledge that we need to capture and represent.*
- *In the second case, specifically, when we consider a software domain, that word represents the technological issues needed to assure the perfect consonance with specific users/experts/customers requirements for the to-develop, in-development or developed software system.*

The Domain Analysis

The Domain Analysis is a comparatively new discipline in the Software Engineering field. *"This discipline defines process to identify and represent the relevant information in a domain, that is, identify and represent a set of systems or subsystems which share common attributes and capabilities"* [1].

Thus, the Analysis of Domain is generally based on the information or logical conclusions that the person can obtain in a certain related group of systems and subsystems in that domain and sometimes information on the goals to be reached. *The information is derived from:*

- The study of existing systems and development histories
- Knowledge captured from domain experts
- Underlying theory
- Emerging technology

Domain Analysis and Requirement Analysis

When a corporation, a company, a governmental agency, a school or another organization wishes to transform systematic people-performed-procedures in software, the first step for this is the system requirement analysis. The

second step pointed by Software Engineering, in these cases, is the software requirement analysis and it is based on the first step. Nevertheless, systematic procedures are very complex and require a software family to perform the desired software performances.

Software requirement analysis is focused specifically on software needs and is based on the performed system requirement analysis, that is, requirements must include restrictions and constraints imposed, for instance, by the hardware to be used, by the time available for development and delivery, by costs.

When the systematic procedures of an organization are complex, needing a family of software products to assure the quality of its work, the Domain Analysis is suggested as the best way to capture the majority of the domain characteristics and requirements.

There is a difference between Requirement Analysis and Domain Analysis:

- *The Domain analysis is "the process of identifying, collecting, organizing, and representing the relevant information in a domain, based upon the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain" [1]. The domain analysis methodologies generally address the formulation of a Domain Model.*
- When we perform a Requirement Analysis, basically what we carefully need to consider is the needs and the demands of the customers or users, that is, the domain does not need to be completely modeled. On the other hand, the domain, in this case, generally is not fully embraced or completely surrounded, because the domain model is accomplished starting from the point of view of these customers and users.

Domain Model

What we need, when we structure a Domain Model, is the precise delineation of the scope of an application for all objects in this domain (or particular sub-domains), pursuing the desired input/output behaviors and performance of the system. Therefore, a Domain Model can be viewed as a representation of a knowledge domain. Particular or restrictive representations of this domain are named as sub-domain representation.

When the identification and collection of the relevant information in a domain or sub-domain is "perfectly done" the model can correspond to a "reality" or to a real world. But, the perfection is not attainable, and the model can be always viewed as an object under construction.

The domain model methodologies can elicit the software modeling, and it is maybe followed by the specification of the software architecture (or an architectonic design) claimed to solve problems in the modeled domain.

ABOUT SOFTWARE ARCHITECTURE

The *Software Modeling* is the search for a solution via software for specific problems in a knowledge domain. The Software Modeling generally guides the Software Architecture design. The Software Architecture is the structure that usually corresponds to a domain for which a software system is being built.

When a Software Model or Software Architecture Model can be shared by a group of software products we have a family of software products.

The definitions of the family of software products, or more precisely, a software product family, vary so much, but the kernel of many of these definitions consider as a family a set of software products when they share a common design or a common set of components. On the other hand, there are definitions that consider as a family, software products according to the technology embodied in them. In all of these definitions there is the concept of a shared kernel. This shared kernel is what many experts define as a software domain.

A RAD Software Architecture Model

In RAD technology, a well-conducted and exhaustive Domain Analysis may be generating a satisfactory Domain Model, and this Domain Model can generate, as consequence, the Software Architecture. When we use RAD systems, the Software Architecture becomes the high-level design, a framework that anticipates the software construction, that is, the architecture represents the design choices that will guide the software development. Wrong choices or forgetfulness made during this phase imply in future damages for the project.

Through the Software Architecture we may show and we can define the basic building blocks for the software.

Layers of a Software Design

The Software architecture represents the first layer of a software design. This first layer can include for instance the following elements: blocks, modules and databases, and also, the interconnections between these elements (a box-and-line diagram) and labels describing its nature.

The second design layer of Software Architecture, indicated for RAD technology, includes the software interface design for the user and, sometimes, an interface for a manager, because box-and-line diagrams are not sufficient as efficient software design documents. It is very important that the second design level map correctly the first design level requirements and be well detailed.

Now, the question is: box-and-line diagrams and the interface design, are sufficient to document a software design when we use the RAD?

Our researches and preliminary results show that these two design layers are sufficient to document the software development, when the adopted methodology is the RAD. We will show this ahead in this paper.

Architectural Style

When a software architecture is consecrated and it becomes adopted freely in many projects we start to have an architectural style.

Architectural Styles generally consist of a generic description in terms of their topology. Modular components, connectors, properties of those modules and connectors, types of interactions, types of data and control, and any constraints on the design, generally are the main elements referred in that description.

A SOFTWARE FAMILY

Nowadays, companies generally develop families of softwares instead of a single software product. A software family is a set of softwares (software products) which can share a kernel or any plug-and-play software elements. In short, software products that perform functions in a specific domain and share parts are considered as a product family. On the other hand, the multiple releases of a single product that include remarkable enhancement can be considered as a product family too, but it is not our case.

Architectural Models that can support product family concerns are very suitable and useful, because it is cost-effective for companies to develop software families rather than single products.

The main problem in developing software product families is to establish software architectures that must, not only address evolution and diversification of products, but also to assess and establish requirements for future developing variant or new products. The kernel of software families must be designed and developed-for-reuse and must support a set of unknown requirements details, that is, usually many of the details are unknown until real products are created.

PEDAGOGICAL DOMAIN

The methodology RAD with the adaptations suggested in this paper was utilized by academic students for the development of educational softwares, under teachers' supervision.

Educational software involves Pedagogic Knowledge, and we developed a domain analysis in the Pedagogic field that could establish a model of software architecture and consequently a model of family of softwares. We explain that idea in the next paragraphs.

Pedagogical Domain and Pedagogical Analysis

A pedagogical domain comprises, at least, four main levels: (1st) the Knowledge Domain; (2nd) the interfaces; (3rd) the Educational Models and (4th) the assessment tools. We will explain here, in the following sub-sections and figures, the main characteristics of these four levels of our Pedagogical Model.

1st Level: The Knowledge Domain

Knowledge is the sum or range of what has been perceived, discovered, or learned. A Knowledge Domain is a particular area of knowledge that a person or groups of people have.

A Knowledge Domain generally is constituted by background knowledge or basic knowledge [2], related to the conceptualization or definition of what is observable in the "Real World", which allows us to build a descriptive model (data model) that approximately corresponds to the real things (objects and artifacts) or abstract things (feelings, beliefs etc), and properties, instantiations, situations or facts about, or involving objects (metadata).

The human knowledge is very abundant and it possesses multiple perspectives. On the other hand, the pedagogic knowledge is a small part of this knowledge, organized in a way to be offered formally through learning opportunities. Besides, the Pedagogic Knowledge is based on the mind and beliefs of educational experts. Usually, the one that is extracted, the specific knowledge, it is a minimum part of the experts' experience.

Another important characteristic of Pedagogical Knowledge is the necessary association of this knowledge with Educational Psychology (see figure 1).

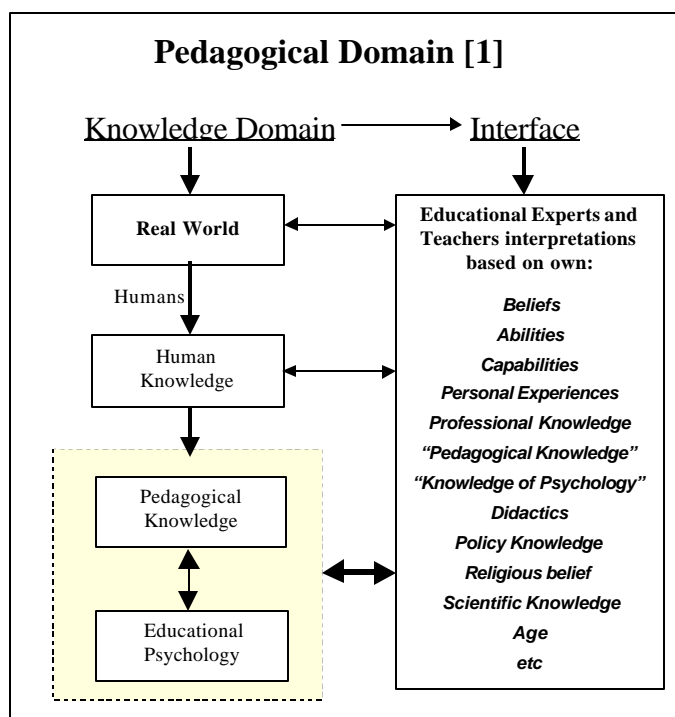


FIGURE 1 - THE PEDAGOGICAL DOMAIN – KNOWLEDGE DOMAIN AND ITS INTERFACE

2nd Level: The Interface

The interface between Pedagogical Knowledge and learners is represented and promoted by: firstly, the educational experts and, secondarily, the teachers. Teachers' beliefs are rooted on there experiences, and are based on several fields: religious, political, scientific, professional etc.

3rd Level: The Educational Models

The prevailing educational models characterize this 3rd level of the Pedagogical Domain. The present educational model prevailing in educational systems in the occidental world is the Constructionism (based on the Constructivism, a notable theory of Jean Piaget).

According Constructionism [3] people construct new knowledge with particular effectiveness when they are engaged in constructing personally-meaningful products, that is, the learning will be more significant and transferable when the students have the opportunity to assimilate the knowledge from their own point of view, predicting and testing their own theories in microworlds and materials or pre-structured objects. In short, Constructionism intends the creation of learning opportunities through: real world problem solving; problem-based learning; cooperative learning; active learning and conceptual adherence test.

Constructionism possesses educational positions quite different from the Behaviorism. However what happens at the schools is the following: *the theoretical speech is based on the Constructionism but the actions are usually based on the Behaviorism*. For this reason, we decide to include some of the ideas of the Behaviorism in our model of Pedagogic Domain (see figure 2).

4th Level: The Assessment Tools

The assessment in behaviorist environments involves: essay questions; interpretive questions; short-answer questions; matching questions; multiple choice questions; true/false questions; homework research-driven (for instance: monographic works followed by oral exhibitions with slides). On the other hand, the assessment in Constructionism environments is based on: self-evaluation; analysis of his/her own scores; peer-comparative or self-comparative evolution and Conceptual Adherence test.

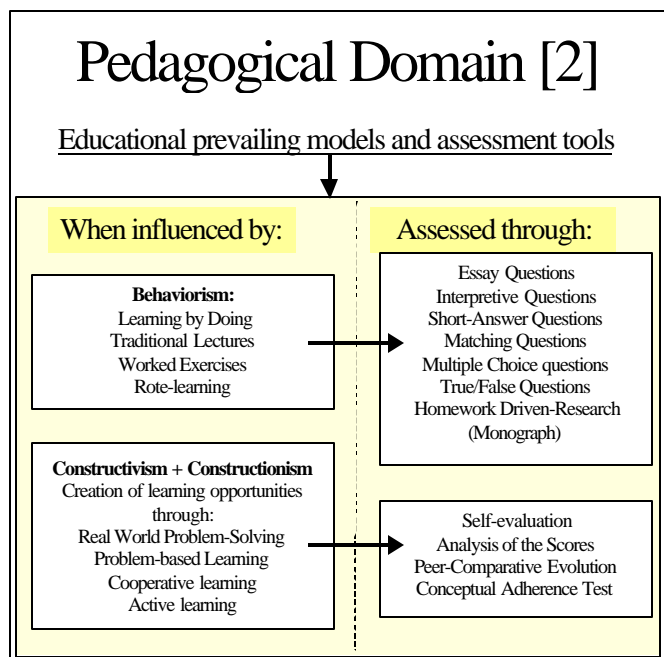


FIGURE 2 - THE PEDAGOGICAL DOMAIN – EDUCATIONAL MODELS AND ASSESSMENT TOOLS

What is Conceptual Adherence?

Before trying to define Conceptual Adherence [4] it is preferable simply suggest some of its characteristics. These characteristics may be summarized as follows:

1. Conceptual adherence is a systematic and interactive process involving two elements: a learner and the most capable peer (a human or a computer).
2. Conceptual Adherence is the process that allows to the most capable peer to evaluate the learner's concept formation (abstraction and generalization) through the comprehension and extension of a required concept.
3. The conceptual adherence only occurs when the learner can really convince the most capable peer about his knowledge upon a specific concept.
4. The levels of a concept comprehension and extension will be flexible to allow a local and instantaneous adaptability, independent of the student model, and to allow the consequent conceptual adherence certifying.
5. When the conceptual adherence does not occur, the most capable peer should start a new process of concept presentation to support the process of adherence to such required concept.
6. When the conceptual adherence occurs the most capable peer can start a new process of concepts presentation or simply interrupt the process.

Defining Conceptual Adherence Test

Conceptual adherence test is a function of two variables: one concept and one computer (or person).

The application of conceptual adherence to these two variables defines the adherence value of a specific person with respect to that concept.

A percentage or a linguistic variable can express these values, for instance: good, medium or bad. For instance, there is no impediment to process these values and compound a profile of this person based on his/her vocabulary.

The definition of conceptual adherence can be expressed by the following functional formula:

$$\begin{aligned}
 &\text{Adherence: (Concept \textcircled{P} People) \textcircled{\{ good, medium, bad \}} \\
 &\qquad \qquad \qquad \textit{when} \\
 &\text{People}=\{ \text{learner, unqualified readers, foreigners, customers,...} \} \\
 &\qquad \qquad \qquad \textit{and, for instance:} \\
 &\text{Adherence}(\text{concept, learner}) \textcircled{\{ value \}}
 \end{aligned}$$

The Students' *Conceptual Adherence* is the key for the students' modeling and, consequently, the Concept Dictionary is the main database that shares information with all elements in software family considered in this work.

A DOMAIN-BASED PEDAGOGICAL SOFTWARE ARCHITECTURE

According to the Pedagogical Model generated via domain analysis methodology, we can establish the following desirable characteristics for educational softwares [5]:

- **They should allow a cooperative triangular association, which involves:** teachers (educators); computers and students; teachers may discuss about any pedagogical *subject matter* (matter under consideration in a written work or speech; a theme) with students when the system suggests or points out;
- **They should be shell systems**, that is, they should admit teachers' intervention, or more exactly, they should be authoring tools;
- **They should involve the Conceptual Adherence** to generate student's conceptualization model.

and

- **They should provide support for the following interventions or actions when teachers access or uses the system:**
 1. insert, adapt, modify or exclude data in all databases;
 2. insert students data and create a password for each student;
 3. perform the text of consistence of all databases – using a validation tool, a Database Validation Tool;
 4. access the information about students' performance and models (Historical and Conceptualization students' models) .

- They should provide support for the following actions when students access or uses the system:
 1. training and managing student's actions and decisions;
 2. solve situations of pedagogical conflict;
 3. explain the solutions or show decision-paths
 4. expose tables, maps, figures, videos etc (multimedia resources) when necessary or at the request of the student;
 5. allows to access information about their performance (scores, peer-comparative or self-comparative instantaneous evaluation or historical performance).

AN ARCHITECTURAL MODEL DESIGN

These ideas: the *software development-for-reuse* (generating a reusable architectural style) and the *learning through cooperative and constructive actions*, naturally point to an original architectural model design, based on the software product family concept.

The software family shown here (see figure 3) is a set of shells which contains the following subsystems:

- Classroom Lecture Presentation - a matrix containing pages of one or more classroom lectures;
- Traditional Tutor - a browser and presenter of questions based on long texts that enables a secure learning by rote process;
- Query Applicator - dedicated to the pedagogical heating intending to model the student, that is, create a instant model;
- Test applicator - an adaptive test applicator based on the student's conceptualization model.

This family shares the following databases:

- Concept Dictionary;
- Query Database ;
- Pages of Classroom Lectures;
- Pedagogical Texts;
- Student's Model;
- Student's Scores;
- Personal Students' Repositories

The following figure shows a box-and-line schema representing the architecture model for an educational software product family, based on conceptual adherence.

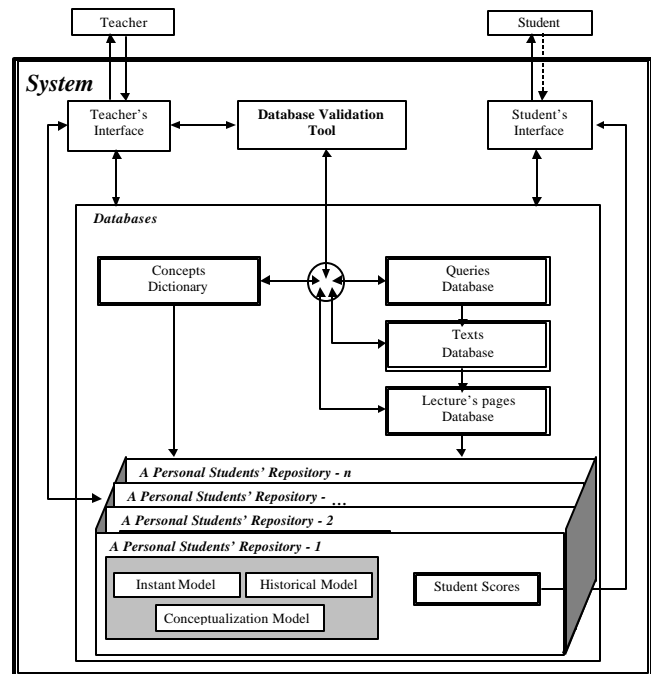


FIGURE 3 - AN ARCHITECTURE MODEL FOR AN EDUCATIONAL SOFTWARE PRODUCT FAMILY BASED ON CONCEPTUAL ADHERENCE

CONCLUSIONS

Secure methodologies and procedures for design and ensuring quality, test, maintenance and distribution of *large software systems* was created, indicated and is in use by Software Engineering Institute/Carnegie Mellon University (SEI/CMU). Many concepts nowadays referred and used by software engineers around the world are easily encountered in technical reports, handbooks and scientific papers available without fee for read and copy, in a very especial site on the Internet which address is: <http://seir.sei.cmu.edu/>. Nevertheless, many of those methodologies and procedures, strongly indicated for *very large software systems development*, are not perfectly compatible with RAD methodology, needing reformulation when one intends to develop small software systems [6].

The Software Architectural-Model here presented, proposed for a small system when we use the RAD methodology, is under construction and test [7], and two elements of this family can already be used for test: the Classroom Lecture Presentation and the Test Applicator. Both include an interface for teachers (the system's administrator) and another interface for the students.

REFERENCES

[1] Hess, H. et al. "A *Domain Analysis Bibliography*", Carnegie Mellon University/Software Engineering Institute - CMN/SEI, URL: <http://www.sei.cmu.edu/publications/documents/90.reports/90.sr.003.html>, Special Report -90-SR-3, last modified: 16 November 2000.

[2] Diederich J.R., Milton J., "Creating domain specific metadata for scientific data and knowledge bases", IEEE Trans., Knowledge Data Engineering 3(4): 421-434, 1991.

[3] Papert, Seymour "THE CHILDREN'S MACHINE - Rethinking School in the Age of the Computer", Basic Books, New York, 1993.

[4] Sá Leite, Aury & Omar, Nizam. ***The Utilization of Conceptual Adherence on the Internet***", SSGRR 2000 Computer & Business Conference – Scuola Superiore G. Reiss Romoli, L'Aquila, Italy, July 31, August 06, 2000, published in CD-ROM.

[5] Omar, Nizan & Sá Leite, Aury. "***The Learning Process Mediated by Intelligent Tutoring Systems and Conceptual Learning***". International Conference on Engineering Education/ICEE 98, Brazil, Rio de Janeiro, 1998, page 20 of abstracts and Session 5 in Proceedings (CD-ROM).

[6] Johnson, Donna L. and G. Brodman, Judith, "***Applying the CMM to Small Organizations and Small Projects***", Proceedings of the 1998 Software Engineering Process Group Conference, Chicago, IL, 9-12 March 1998.

[7] Sá Leite, Aury & Omar, Nizam. "***Combining Artificial Intelligence and Human Problem Solving: Proposing a New Architecture***". In: Artificial Intelligence in Education – Open Learning Environments: New Computational Technologies to Support Learning, Exploration, and Collaboration; S.P. Lajoie & M. Vivet (Eds). Burke, VA, USA; IOS Press, 1999. Listed as a poster on conference schedule. AI-ED'99 – 9th International Conference on Artificial Intelligence in Education – 19/23 July 1999 – Le Mans, France.