ABSTRACT

Problem-Based Learning Situations require accurate template models in which the roles of tutor and learner participate in various modified cooperative activities. This paper discusses the use of the UML to first build such customizable models, and next to derive Educational Software Components from models. The paper contributes to reduce the lack of flexibility in open distance learning tools where distribution of components applies with some difficulty. It is on purpose introduced the designer role for problem-based learning situations. This designer aims to assemble educational components in order to offer computer-aided learning supports. Model examples and techniques to implement components are also briefly evoked.

Keywords: Educational Component, Cooperative Learning, Problem-Based Learning Situation, UML.

1 INTRODUCTION

Our work copes with learning situations involving cooperation between tutors and learners. Within this context, computer-aided learning hinges on software and platforms whose customization allows to implement scenarios embodying such cooperation. Because of the monolithic aspect of learning platforms and software, as well as the specificity of education based on Problem-Based Learning Situations (PBLs) (Meirieu, 1988), we propose in this paper an approach using the notion of Educational Software Component (Roschelle et al., 1999). PBLs rely on cognitive models of cooperative activities for tutors and learners. These cognitive models can be specified once for all and captured within components. By offering enough flexibility, namely parameterization to keep a good degree of tuning, and by naturally supporting distribution, Educational Components, when reused, allow to assemble new PBLs in software systems. Distance learning issues via distribution are especially associated with the idea of software component.

We focus in this paper on the UML specification of Educational Components. This formalism favors, at a conceptual level, the description of tutor/learner and learner/learner cooperation. Within this context, computer-aided learning hinges on software and platforms whose customization allows to implement scenarios embodying such cooperation. Because of the monolithic aspect of learning platforms and software, as well as the specificity of education based on Problem-Based Learning Situations (PBLs) (Meirieu, 1988), we propose in this paper an approach using the notion of Educational Software Component (Roschelle et al., 1999). PBLs rely on cognitive models of cooperative activities for tutors and learners. These cognitive models can be specified once for all and captured within components. By offering enough flexibility, namely parameterization to keep a good degree of tuning, and by naturally supporting distribution, Educational Components, when reused, allow to assemble new PBLs in software systems. Distance learning issues via distribution are especially associated with the idea of software component.

We focus in this paper on the UML specification of Educational Components. This formalism favors, at a conceptual level, the description of tutor/learner and learner/learner cooperation. At implementation time, UML supplies Component & Deployment Diagrams to package and deploy specification pieces into components. We sketch in this paper such cooperation and briefly discuss at the end of the paper, implementation based on a dedicated library. Indeed, we divide the modeling of pedagogical activities into Statechart Diagrams and therefore illustrate how to easily and quickly implement these dynamical models in Java.

2 EDUCATIONAL ENGINEERING

Educational Engineering covers techniques and tools that assist, and possibly automate in software, the universal and dual actions of teaching and learning. In this section, after describing our context of work and goals, we walk through current innovative projects in this domain.

2.1 Context of Work

Educational Components here described appear within the framework of a more general project: the specification of an environment allowing a teacher to implement cooperative learning situations. Our first goal is to help teachers to specify learning situations. PBLs are quite different from classic learning approaches based on the notions of courses, exercises, assessments. PBLs are indeed based on the idea of cooperative activity and more exactly cooperative resolution of problems.

Recent works (Nodenot et al., 2002) describe actors, and application principles of such learning activities. We pay attention on the definition of a system (called Learning Management System or LMS) that manages activities for distant user communities (learners and tutors). We also show that a pedagogy relying on cooperative activities conforms to normalization works carried out by international e-Learning consortia (AICC, 2000; ARIADNE, 2001; Dublin Core, 2000; GESTALT, 2000; IMS, 2000; ISO/IEC JTC1/SC36/ WG2, 2001; LTSC, 2000; MASIE_Center, 2002; PROMETEUS, 2000; SCORM, 2000).

Previously, we worked on the specification of a role-component library enabling a designer to reuse learning scenario (Sallaberry et al., 2002). This approach allows the assembling of new roles by combining pre-existent role-components.

2.2 Educational Components

Over years, research in Educational Engineering emphasizes the support of interoperable and reusable applications as well as “electronic” services (Wiley, 2000). The Educational Component (EC) approach is growing: “having component developers collaborate with domain experts to build applications may be the future of software development” (Roschelle et al., 1999).

The component paradigm used within the Educational domain has numerous objectives:

1. sharing learning resources between software tools and systems,
2. making interoperability between tools,
3. making interoperability between applications and learning resources,
4. reusing learning resources (by teachers),
5. reusing educational software (by developers).

As the word “Component” in Software Engineering, “Educational Component” has multiple meanings and may have very different interpretations. Thus, we cannot find out a generic definition of an EC. We point out the dual notion of Learning Object (LO). LOs are “any digital resource that can be reused to support learning” (Wiley, 2000) or “Learning Objects are defined here as any entity, digital or non-digital, which can be used, re-used or referenced during technology supported learning” (LOM, 2000). In fact, such a fine-grained component supports learning by means of its embedded learning content. So, research works on LOs rather correspond to points 1-3-4 above (ARIADNE, 2001; De_La_Passardière et al., 2001; IMS, 2000; Koper, 2001; LOM, 2000; SCORM, 2000) and relate to classical learning concepts as courses, exercises, assessments. LO approach is more convenient for the common learning approach than for the PBLS constructivist one (Deschênes et al., 1996) – learner build his own knowledge by doing.
2.1 Current Trends and Directions for Educational Software Components

We study different projects based on Educational Software Components that we split into two categories.

ESCOT (Educational Software Components Of Tomorrow) project (ESCOT, 2002) aims at the construction of a digital library containing educational software relating to Middle School mathematics. One of the ESCOT’s goals is to have interactive JavaBeans-based content within an educational context. In the same way, ESCOT explores the processes of distributed software-development with the specific objective to rapidly build and deploy reliable software (Repenning et al., 2001). It also deals with the EC stemming from AgentSheets (Agent_Sheets, 2001) and E-Slate projects (Birbilis et al., 2000). AgentSheets is an authoring-tool for the creation of reusable EC under the shape of applets, directly integrable on a Web page. These components can be used for exercises of simulation, demonstration, scientific modeling, etc. The technology used is simple and the integration on a Web page implies that the component is only a pedagogical element of a bigger one. In E-Slate, components are supplied in the form of prefabricated objects (card, clock, vector…). They possess a mechanism of interconnection (glue) and are configurable, customizable: they can contain features for a specific domain. They are connected together within the same global application which allows to visually assemble them according to “the puzzle’s analogy”; they especially have an appropriate graphical interface.

The works on SimulNet (Anido et al., 2001) propose a layered component model for the support of Web-based interactive and collaborative applications (framework) as well as a development of an educational application based on this model. Every component aims at supplying a feature required in Web-based collaborative applications. In a similar way, the global objective of the PLACE project (PLAteforme à Composants Evolutif) (Peter et al., 2002) is the following study: how to realize flexible cooperative working environments based on models of standard components. In both SimulNet and PLACE projects, the Software Component principles are applied in the design and the realization phases: the creation of a Web environment dedicated to distant learning. For the PLACE project, the architecture is based on the use of the EJB / J2EE platform in order to build a CSCW (Computer Supported Cooperative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture. At the structure level, the analogy between component and tool (auditingenerative Work) platform whereas SimulNet proposes a client / server architecture.
This pedagogical activity concerns two actors: a learner and a tutor. Thus, the pedagogical activity representation requires two Statecharts for each possible pedagogical treatments linked to one of the actors (Figure 1). The required precision for this component configuration is: who are the tutor and learner? The answer will enable the instantiation of the generic parameters “Tutor” and “Learner”. During the design phase, the configuration may also concern the types of the messages: synchronous (instant messaging) or asynchronous (email). Moreover, this figure shows two different kinds of events: “need help” that corresponds to an event generated by the user “Learner”; “received help” (gray state) that corresponds to an independent event of the “Learner”, it is generated by the LMS or another user (“Tutor”). Every transition involves an action-element of this shape: send (Target, Method, Parameters ) - method objects correspond to services provided by the target.

The following figure shows an example of assembly between the concern learner part (“Yet designed Learner Side”) and a statechart (“Yet designed Pedagogical Activity”) representing the pedagogical activity predicted by the PBLS designer. This second statechart is deliberately limited to one and only state (“in_progress”) in order to hide pedagogical activity complexity. Both statecharts are assembled by aggregation into a new global state (“My Final Pedagogical Activity”). It means that both sub-statecharts can evolve separately during the actual realization of the pedagogical activity: while following sub-statechart activity, the learner may request some help and wait for the tutor answer. This “concurrent” mode is a pedagogical assembly possibility but other alternatives remain as direct chaining of two statecharts.

3.4 Implementations

We sketch here how our statecharts are implemented in order to become operational components. Here, the statechart of Figure 2 (“My final Pedagogical Activity”) is build and execute in the same way as a simulation-tool in order to validate our model. To this end, we use the PauWare Statechart Java library.

```java
protected Tutor _tutor;
protected Statechart _interruption;
protected Statechart _S1;
protected Statechart _S2;
protected Statechart_monitor _My_final_pedagogical_activity;

_My_final_pedagogical_activity = new Statechart_monitor(_S1.and(_S2).xor(_interruption));
// _S1 & _S2 states are concurrent à "and" assembling
// "interrupted" state is in exclusive or with the assembled _S1 & _S2 à "xor" assembling
_events influence and conduct the way by which learning activities may run:

synchronized public void need_help() {
    try {
        // transition "need help" build between states from "no_need" to "awaiting_help_response"
        _My_final_pedagogical_activityfires(_S1.and(_S2).xor(_interruption));
        System.err.println("Send question", no_need);
        _My_final_pedagogical_activity.Used_up();
    } catch(StatechartException se) {
        System.err.println("CMI Guidelines for Interoperability v3.4; released [WWW Document].
        retrieved from the World Wide Web: [WWW Document].
        Copyright © 2003, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.
```

4 CONCLUSION

In this paper, we introduce the idea of Educational Component in order to support computer-aided pedagogical activities. We review some current research works on this special concept of Educational Component. Our approach converges towards the modeling of Educational Components with the UML as well as the potential associated implementation of these components. Capturing learning/teaching activities that are in essence cognitive processes, comes up against inappropriate formalisms. We think that a suitable formalism is well as the potential associated implementation of these components. Capturing learning/teaching activities that are in essence cognitive processes, comes up against inappropriate formalisms. We think that a suitable formalism is.


Footnotes

1 A re-routed activity is an individual pedagogical activity dynamically allocated; this allows the system to propose new pedagogical activities every time a user finishes an activity and is waiting for other ones, avoiding passive delay.