A System Architecture for Affective Meta Intelligent Tutoring Systems

Javier Gonzalez-Sanchez[1], Maria Elena Chavez-Echeagaray[1], Kurt VanLehn[1], Winslow Burleson[1], Sylvie Girard[2], Yoalli Hidalgo-Pontet[1], and Lishan Zhang[1]

[1] Arizona State University, Tempe, AZ, USA
{javiergs,mchaveze,kurt.vanlehn,winslow.burleson, yoalli.hidalgopontet,lishan.zhang}@asu.edu
[2] University of Birmingham, Birmingham, UK
s.a.girard@bham.ac.uk

Abstract. Intelligent Tutoring Systems (ITSs) constitute an alternative to expert human tutors, providing direct customized instruction and feedback to students. ITSs could positively impact education if adopted on a large scale, but doing that requires tools to enable their mass production. This circumstance is the key motivation for this work. We present a component-based approach for a system architecture for ITSs equipped with meta-tutoring and affective capabilities. We elicited the requirements that those systems might address and created a system architecture that models their structure and behavior to drive development efforts. Our experience applying the architecture in the incremental implementation of a four-year project is discussed.

Keywords: architecture, component-based, tutoring, meta-tutoring, affect.

1 Introduction

Intelligent Tutoring Systems (ITSs) seem capable of becoming untiring and economical alternatives to expert human tutors. This possibility has proven difficult to achieve, but significant progress has been made. The use of ITSs has become more common, and there is significant work about their pedagogical and instructional design but not about their technological implementation. ITSs are software products and, as for any other software product, their implementation on a massive scale relies on the principle of assembly instead of crafting them as one-of-a-kind systems. Component-based software engineering [1] is an appropriate approach for handling mass production. Component-based software engineering addresses the development of systems as an assembly of parts (components), with the development of these parts as reusable entities and with the maintenance and upgrading of systems through customizing and replacing such parts.

Following a component-based approach, we have defined a system architecture to drive the development of ITSs equipped with affective and meta-tutoring capabilities, called affective meta intelligent tutoring systems (AMTs). Defining a system architecture is the first step in creating a component-based software framework to implement AMT-like applications. This system architecture takes advantage of previous experiences with ITS implementations; most of that previous experience was extracted from the analysis made on existing ITS behavior described in [2], as well as from previous experience in the development of real-time affective companions, mainly by the work described in [3].

This paper is organized as follows: Section 2 provides the terminology and background for system architectures, ITS behavior, and affect recognition; Section 3 describes the AMT system architecture; Section 4 describes the implementation of an application following the AMT system architecture and discusses its software metrics; and Section 5 provides a conclusion.

2 Terminology and Background

The following terminology and background summary contextualizes the work described in this paper:

System Architecture. A system is a group of interacting, interrelated or independent modules forming a complex whole. Modules are self-contained entities that carry out a specific function; they are implemented as a set of parts called components. A system architecture is a conceptual model that describes the modules and components of a system and how they interconnect with each other; it becomes a software design model by mapping each component to a set of classes following software engineering methodologies. The system architecture is essential for realizing the system's quality attributes [4].

ITS Behavioral Description. ITSs are typically used to assign tasks to students; tasks are composed of steps that the student must accomplish. The structure of this kind of ITSs, called step-based, is described in [2] and can be summarized as follows: (1) the group of tasks known by the ITS conforms its *Knowledge Base*; (2) a *Task Selector* chooses from the *Knowledge Base* the Task that the student must solve by considering the student's previous performance reported by an *Assessor*; (3) a *User Interface* (a tool or an environment) provides the space in which the interaction between the tutor and the student occurs; (4) a *Step Analyzer* methodically examines the student's steps and determines whether they are correct or incorrect and then reports that information to a *Pedagogical Module* and to an *Assessor*; (5) a *Pedagogical Module* provides support (hints and feedback); the provided support depends on current steps and the student's previous performance; and (6) an *Assessor* measures the performance of the student (requested hints, time used to go from one step to another, etc.).

Affect Recognition Strategies. Research shows that learning is enhanced when affective support is present [5]. To provide that support, ITSs need to recognize students' affect. Diverse strategies exist for affect recognition; the one we are considering for this work uses sensing devices to read students' physiological responses; this strategy uses, among others, brain-computer interfaces, eye-tracking systems, face-based emotion recognition systems, and diverse sensors to measure skin conductance (arousal), posture, and finger pressure [6].

3 System Architecture

The system architecture was engineered [7] on the principles of encapsulation, low coupling, centralized shared data, and layering. Functionality is encapsulated in simple components; components that are complex and/or serve diverse purposes are split into several collaborative components. Components are low-coupled to facilitate replacement, i.e., to increase modifiability. A centralized data-sharing mechanism is used to pass data among modules to reduce latency. Components are organized in a three-layer structure in which the bottom layer encodes utility services for data management and communication responsibilities; the middle layer encodes the business logic; and the topmost layer encloses the user interface, which handles the interaction with the user. Since the user interface is particular to a specific system, it is not described here. Fig. 1 shows modules (boxes), components (gray boxes), and their relationships (arrows) as follows:

Tutor Module. It encapsulates the ITS behavior. Its components and relationships are summarized in Section 2.

Meta Tutor Module. It encapsulates the logic for providing meta-tutoring recommendations and promoting meta-skills in the student. The *Meta Tutor* module has two components: (1) an *Inspector* that reads *Tutor* events (populated in the *Shared Repository*) and filters those that suggest an intervention is needed; and (2) an *Engine* that provides intelligence to the *Meta Tutor*; the *Engine* is notified by the *Inspector* of compelling events and it infers the type of intervention that must be done. Interventions consist of showing a message or disabling channels of user interaction. The *Engine* implements the policies about how and when interventions must be done. It communicates the interventions to the *User Interface* for its execution.

Affective Companion Module. It encapsulates the logic for generating affective interventions. The *Affective Companion* has two components: (1) an *Event Selector* reads the data for *Tutor* events and affective states (populated in the *Shared Repository*) and filters combinations that suggest an intervention is needed; and (2) an *Affective Engine* that implements the affective intelligence; the *Affective Engine* is notified by the *Event Selector* of compelling combinations of *Tutor* events and affective state data and infers the type of intervention that must be done. Interventions consist of motivational messages. The *Affective Engine* implements the policies about how and when interventions must be done. It communicates the interventions to the *User Interface* for its execution.

Shared Repository. It is a centralized means for passing data among the other modules, which are running concurrently. The *Shared Repository* module follows a blackboard architectural model, in which a common data repository, “the blackboard,” is updated by some modules and read by others. The *Tutor* posts events to the blackboard and the *Emotion Recognition Subsystem* posts affective state reports. The *Meta Tutor* and *Affective Companion* observe the blackboard, looking for data that triggers an action on their side.

Emotion Recognition Subsystem. It is a facade that provides a simplified interface to a source of affective state data, such as a third-party system, framework, or library.

The system architecture prioritizes the quality attributes of modifiability, extensibility, and integrability. Modifiability refers to the ease with which a component can be modified for use in applications or environments other than those for which it was specifically designed; affective and cognitive intelligence require this quality since they are implemented in different ways. Extensibility refers to being prepared for extension into unforeseen contexts since not all application requirements can be determined in advance; our system architecture required this quality to make feasible the addition of new tutoring, meta-tutoring, or affective support capabilities. Integrability is the process of combining software subsystems to assemble an overall system; AMT system architecture requires the integration of a third-party system or code (1) for affect recognition to support the functionality of the *Affective Companion* module and (2) for decision-making (machine-learning algorithm implementation) to support the functionality of the *Affective Companion* and *Meta Tutor* modules.

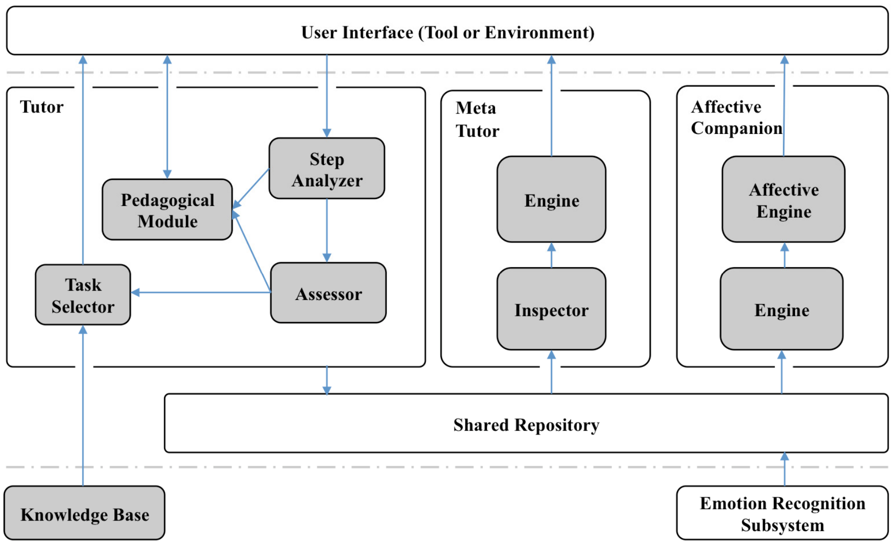

Fig. 1. AMT System Architecture

4 Usage and Discussion

The AMT system architecture has been used as a reference during a four-year project focused on developing an AMT application [8]. The AMT application was implemented in Java with Swing components. The final version is composed of 16 packages, 120 classes, 1507 methods, 1810 attributes, and 37,374 lines of code. A production-rule system and a third-party implementation of emotion recognition algorithms were used to support the application development. A detailed description of moving from AMT system architecture to software design is outside the scope of this paper due to space limitations; nevertheless, a description of mapping the ITS module to a software design can be found in [9]. The four-year implementation process was managed using a revision control system and comprises 1,643 revisions and 8 released versions. Differences between released versions include, among others, changes in requirements, enhancements of decision-making strategies, and bug fixing. A total of 15 developers were involved in the different stages of the project, and a team of at least four developers was working concurrently in every stage.

The results of applying the system architecture were measured indirectly by evaluating the structural software quality of the systems developed under its influence using software metrics. Due to space limitations we report the evaluation of four AMT application releases, one from each development year, as follows: (1) Release 742 implemented the first deployed *Tutor*; it was focused on the *User Interface* (a tool) and had limited tutoring capabilities; coding the skeleton of the system was the primary goal during this year. (2) Release 1277 refashioned the *User Interface* and implemented an enhanced *Tutor*. (3) Release 1545 included a *Meta Tutor*, continued refashioning the *User Interface,* and enhanced the *Tutor* module. (4) Release 1643 added the *Affective Companion* capabilities, enhanced the *Meta Tutor*, and refactored the *User Interface* and *Tutor*. The metrication of structural qualities, shown in Table 1, includes measures for size, complexity, and coupling as follows: number of packages (P), number of classes (F), number of functions (Fn), number of lines of code (LoC), number of comments (LoCm), average cyclomatic complexity (AvC), maximum afferent coupling (MaxAC), and maximum efferent coupling (MaxEC) [10].

Table 1. Comparison of software metrics for modules in diverse AMT application releases

Release	Tutor								
	Date	P	F	Fn	LoC	LoCm	AvgC	MaxAC	MaxEC
742	07/2010	5	24	347	10656	2861	3.11	4	5
1277	07/2011	9	42	650	20839	4127	3.61	8	9
1545	07/2012	11	55	885	24542	4654	3.03	9	9
1643	07/2013	14	62	936	25189	4816	2.96	12	10

Release	Meta Tutor								
	Date	P	F	Fn	LoC	LoCm	AvgC	MaxAC	MaxEC
1545	07/2012	1	22	202	3346	437	2.68	4	4
1643	07/2013	1	22	248	4210	458	3.05	5	7

Release	Affective Companion								
	Date	P	F	Fn	LoC	LoCm	AvgC	MaxAC	MaxEC
1643	07/2013	3	36	323	7975	1403	2.59	9	6

Even though we had a high turnover in the development team, the size, complexity, and coupling remained at acceptable values. Size measurements (P, F, Fn, LoC, and LoCm) show a correspondence of the requirements implemented in each release and the size of the application, as well as a balance in its granularity. The average complexity (AvgC) at the module level remains within acceptable ranges (below five); at a fine-grain level (classes), not shown in the table, those values are not always acceptable. The decrease in average complexity in the latest versions of *Tutor* shows the refactoring outcome (functionality was fixed and developers focused on code improvements). Lower values in coupling measures (MaxAC and MaxEC) are better since they are a sign of independence; the high values of coupling in *Tutor* can be justified because they belong to the *User Interface* (highly connected); *Meta Tutor* values are acceptable, but *Affective Companion* values suggest that a refactoring would be required in the implementation of this module.

5 Conclusions

In this paper, we have presented the AMT system architecture, the first step for creating a component-based software framework to implement AMT-like applications. We have defined its requirements and qualities and have shown how the AMT system architecture addresses them to support large-scale reuse. Software metrics for different releases of one AMT application show how the system architecture provided a flexible partition of the system that facilitates modifiability, extensibility, and integrability. With this proposed system architecture, we aim to share our experience, looking forward to making the development of AMT-like systems an easier, faster, and standardized process.

Acknowledgments. This material is based upon work supported by the National Science Foundation under Grant No. 0910221.

References

1. Crnkovic, I.: Component-Based Software Engineering - New Challenges in Software Development. Software Focus 2(4), 127–133 (2001)
2. VanLehn, K.: The Behavior of Tutoring Systems. International Journal of Artificial Intelligence in Education 16(3), 227–265 (2006)
3. Burleson, W.: Affective Learning Companions: Strategies for Empathetic Agents with Real-Time Multimodal Affective Sensing to Foster Meta-Cognitive Approaches to Learning, Motivation, and Perseverance. MIT PhD thesis (2006)
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison-Wesley, Boston (2003)
5. Lehman, B., D'Mello, S.K., Person, N.: All Alone with Your Emotions. In: 9th International Conference on Intelligent Tutoring Systems. Springer (2008)
6. Gonzalez-Sanchez, J., Chavez-Echeagaray, M.E., Atkinson, R., Burleson, W.: Multimodal Detection of Affective States: A Roadmap Through Diverse Technologies. In: ACM SIGCHI Conference on Human Factors in Computing Systems. ACM (2014)
7. Firesmith, D.G., Capell, P., Falkentha, D., Hammons, C.B., Latimer IV, D.T., Merendino, T.: The Method Framework for Engineering System Architectures. CRC Press (2008)
8. VanLehn, K., Burleson, W., Girard, S., Chavez- Echeagaray, M.E., Gonzalez-Sanchez, J., Hidalgo-Pontet, Y., Zhang, L.: The Affective Meta-Tutoring project: Lessons Learned. In: 15th International Conference on Intelligent Tutoring Systems. Springer (2014)
9. Gonzalez-Sanchez, J., Chavez-Echeagaray, M.E., VanLehn, K., Burleson, W.: From Behavioral Description to A Pattern-Based Model for Intelligent Tutoring Systems. In: 18th International Conference on Pattern Languages of Programs. ACM (2011)
10. Pressman, R.S.: Software Engineering, 7th edn. McGraw-Hill, Boston (2009)