E-learning by Doing with Computational Logic

Federico Chesani, Anna Ciampolini, Paola Mello

DEIS – Dipartimento di Elettronica, Informatica e Sistemistica
Facoltà di Ingegneria – Università di Bologna
viale Risorgimento 2, 40136 – Bologna
{federico.chesani, anna.ciampolini, paola.mello}@unibo.it

Abstract

E-learning by doing is an important e-learning process, that provides several advantages but that requires a high interactivity degree, not always supported in e-learning contexts. In this paper, we propose to exploit a formal approach (based on Computational Logic) to define, verify and support the e-learning by doing paradigm. To this end, we introduce the SCIFF formal frameworks, its features, and two software components for e-learning of software applications.

Keywords: Logic Programming, E-Learning by doing, Interaction Protocols, Software Application.

1. Introduction

One of the most promising learning paradigms is the *learning by doing*, where the student directly practices a learning topic onto a real system, or a model that simulates the real system. The student is presented with a problem, whose solution requires the use of the acquired knowledge. By directly practising on the real system, the student enforces such knowledge; moreover, if the solution of the problem has not been given before (and it is not a naïve consequence of the knowledge itself), the student is forced to construct its own solution, hence exploiting and practising the acquired competences. The *learning by doing* approach can be applied also to e-learning processes (Baldoni et al., 2003), and in particular to software applications learning. A further advantage of this approach derives from the fact that many software systems allow different solutions for the same problem: in the e-*learning by doing* there is no need to overload the student with such information, but rather she is left the task of "discovering" such possibilities, making the learning process a personal experience.

The advantages of the *learning by doing* are a consequence of the high degree of interactivity that such model envisages. Supporting interactivity then is fundamental, and weaknesses on this aspect deeply hinder the approach. E.g., the student should receive help and feedback whenever it is opportune, hence avoiding the risk of being blocked. To this end, run-time evaluation is fundamental to automatically provide suggestions to the student, while "a posteriori" evaluation is needed to assess the acquired skills. Moreover, the common situation where the same learning goal can be achieved in more than one way requires the tutoring system to be able to evaluate all the options, and in particular to dynamically adapt to the student choices.

In this paper we present our approach, based on computational logic, to the e-learning by doing model for software applications. We draw inspiration from our previous work on software Multi Agent Systems and agent societies. In particular, within the European project

SOCS[1], we developed a comprehensive framework, namely *S*CIFF, containing theories, languages and tools for defining, constraining and evaluating the observed behaviour of agents in a social context (Alberti et al., 2008). Here, we show how the *S*CIFF declarative language can be used to define learning goals and to rule the learning activities. Then, the same tools used for agent verification can be adopted to perform evaluation at run-time/posteriori, and to provide hints. W have focused our attention on office applications, namely the OpenOffice Suite and the MSOffice 2007, and we have developed a software prototype supporting these two frameworks. The software comes as a plug-in that receives as input the high-level description of a learning activity, and the actions performed by the student. Evaluation or suggestions then are given as output, supporting the interaction required by the learning model.

2. The SCIFF approach to Agent Interaction Protocols

The *S*CIFF language (Alberti et al., 2008) was originally introduced for the specification of global interaction protocols in open agent societies. It is focused on the observable events which occur within an interaction among two or more agents, where with the term "agent" we generally mean a software component, as well as a human user or a robot. We assume the reader to be familiar with Computational Logic, and with Logic Programming in particular; the non accustomed reader can find a good introduction in (Lloyd, 1987).

*S*CIFF models the occurrence of an event *Ev* at a certain time *T* with the predicate *H(Ev,T)*, where *Ev* is a logic programming term and *T* is an integer, representing the discrete time point at which the event happened (the *H* stand for ``Happened''). Beside the explicit representation of what has already happened, *S*CIFF introduces the concept of "what" is expected to happen, and "when". Thanks to the notion of expectation, *S*CIFF allows to specify interaction protocols in terms of rules of the form "if A happened, then B should be expected to happen". *S*CIFF pays particular attention to the openness of interaction: interacting peers are not completely constrained, but they enjoy some freedom. This means that the prohibition of a certain event should be explicitly expressed in the model: to this end *S*CIFF supports also the concept of negative expectations (i.e. of what is expected not to happen).

Positive expectations about events come with form *E(Ev, T)*, where *Ev* and *T* could be variables, or they could be grounded to a particular (partially specified) term or value respectively. Constraints (à la Constraint Logic Programming, (Jaffar et Maher, 1994.)), like *T>10,* can be specified; attaching this constraint on the above expectation means that the expectation is about an event to happen at a time greater than 10. Conversely, negative expectations about events come with form *EN(Ev, T)*; writing $EN(Ev, T) \wedge T>10$ means that *Ev* is forbidden at any time which is greater than 10. Social Integrity Constraints are forward rules used to link happened events and expectations, to the end of defining allowed interactions by means of declarative rules. They come with the form $body \rightarrow head$, where *body* can contain (a conjunction of) happened events and expectations, and *head* can contain (a disjunction of conjunctions of) positive and negative expectations.

The operational counterpart of the language, namely the *S*CIFF proof procedure, is able to verify conformance of a set of interacting entities w.r.t. the considered protocol by hypothesizing positive (resp. negative) expectations and checking whether a matching happened event actually exists (resp. does not exist).

[1] "SOCS: Societies Of ComputeeS" project, IST-2001-52530, 5th Frame Program.

2.1. Applying the SCIFF framework to the E-learning scenario

We start by considering both the student and the real system (a software) as two distinct agents interacting with each other. Each event corresponds to one observable action that the student performs, i.e. mouse clicks, shortcut key pressions, and similar. The *S*CIFF language then allows defining a possible exercise in terms of a goal and of a set of rules the student must follow. Each rule defines the expected, future behaviour of the student, i.e. which are the actions she is expected to do.

For example, consider the following exercise: "The student should create a new document, insert a title, applying the style Heading1 and finally save such document". The goal of the exercise is given by the conjunction of many sub-goals, each one achievable in many different ways. For example, a *S*CIFF rule for the creation of a new document would be:

$$\begin{aligned} H(\text{start}(\text{exercise}), T_s) \rightarrow\ & E(\text{fileMenu}(\text{createNew}), T_c) \\ & \vee E(\text{keyPression}(\text{ctrlN}, T_p) \\ & \vee E(\text{keyPression}(\text{altFN}, T_a). \end{aligned}$$

The right part of the rule contains, in a disjunction, the alternative actions that satisfy the goal of getting a new document. The rule is satisfied if the student performs at least one of the expected actions.

The *S*CIFF Proof Procedure can be exploited then in two different ways. A posteriori, it can be used to evaluate the sequence of performed actions, and determine a score of how much

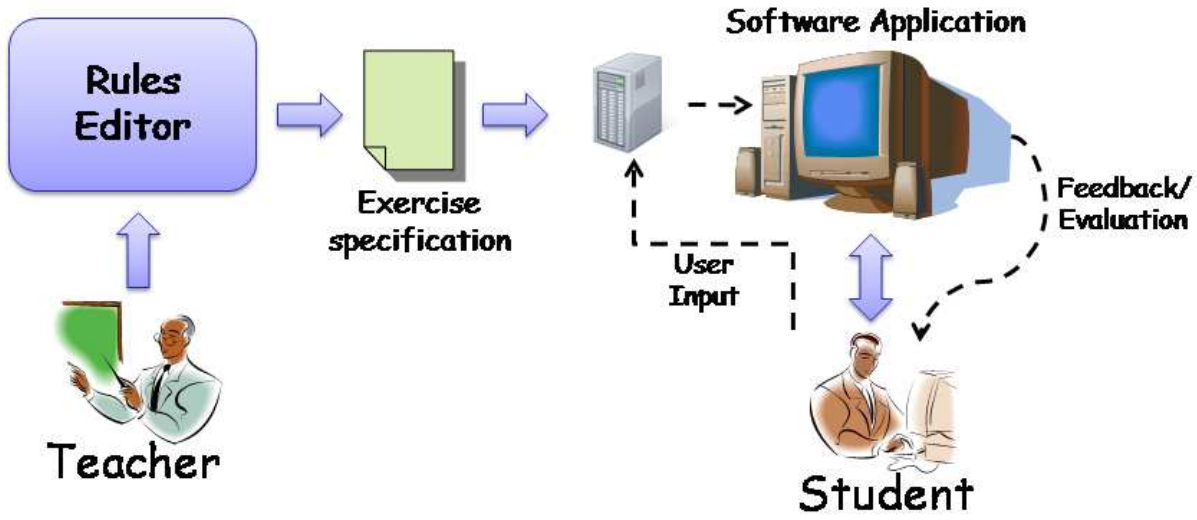

Figure 1: Overall architecture of the SCIFF Verification Plug-in

she has fulfilled the expectations. Such expectations could have been a direct consequence of the exercise goal, as well as they could have been generated by the triggering of some rule.

At run-time, the *S*CIFF proof procedure can be exploited again for evaluation purposes or, more interestingly, to compute, at each moment, which are the expectation about the student behaviour. If needed, the proof procedure can directly provide suggestions. The choice of whether providing a suggestion or not can be defined by the teacher by means of SCIFF rules again.

3. The *S*CIFF Verification plug-in and the Rule Editor

The overall architecture we propose is depicted in Figure 1. The first step consists on defining the exercise, by providing a goal and a set of rules, in terms of the *S*CIFF language. To this end, we developed a visual editor that allows the teacher to abstract from the *S*CIFF syntax (Figure 2). The editor presents to the teacher the list of the available actions the student can perform: the teacher is left only the task of creating a link between a set of actions, and what the student is expected to do if she performs such actions.

The exercise specification in *S*CIFF then is provided as input to the Verification plug-in, whose main task consists on monitoring the student actions, and to provide feedback/evaluation if needed. The Verificaton plug-in then comes with four different components, (1) a toolbar to activate/deactivate and load the exercise; (2) a module (not

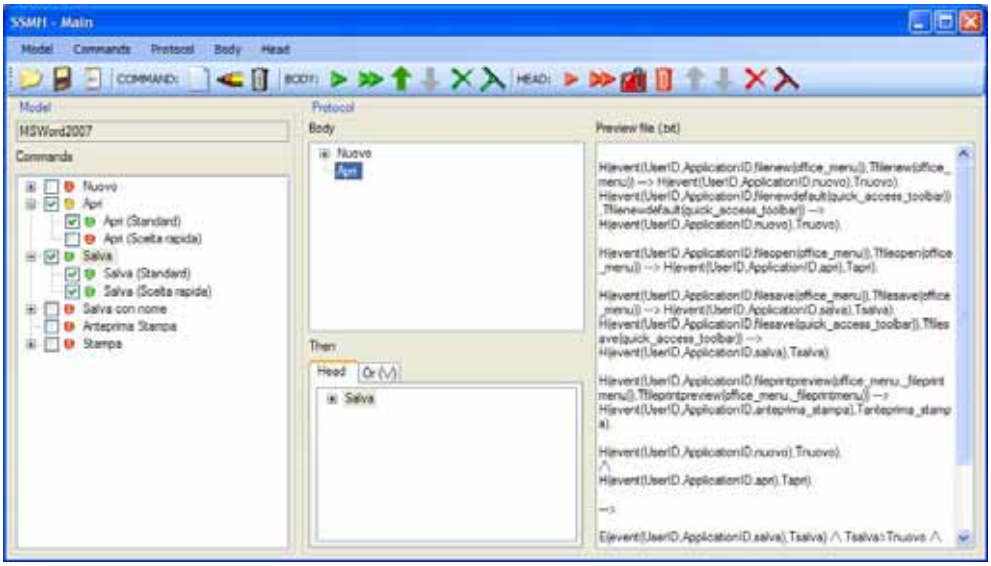

Figure 2: The Rule Editor

visible to the user) to intercepts the student actions and passes them to the *S*CIFF proof procedure; (3) a module encapsulating the proof procedure to evaluate the student behaviour; and (4) a special window where messages can be displayed to the student.

4. Conclusions and Future Works

E-learning by doing is an important learning model that provides several advantages, and in particular the possibility of having students practising learning topics directly on real systems, hence making the learning experience a personalized process. In order to be successful, e-learning by doing model requires a high degree of interactivity; moreover, systems supporting such model must be able to dynamically adapt to a number of different conditions and situations.

Starting from our previous experience on Multi Agent Systems, we exploited the framework *S*CIFF in two ways: as a specification language, to support the definition of exercises in terms of logic programming goals and rules; as a verification tool, to verify if the behaviour of the student respect the constraints given by the exercise. The advantages are manifolds: the use of a declarative language makes easier the specification of an exercise; evaluation can be performed a posteriori, o directly at run-time; moreover, the same tool used can be used to provide feedback, if the student needs it, hence supporting a certain degree of interaction.

The developed software components, i.e., the Rule Editor and the SCIFF Verification Plug-in, are yet in a prototypical stage. We have started some testing with some students, getting some

positive feedbacks but also some remarks about the feasibility of specifying complex exercises. A far deep testing of our approach is needed before we can draw any conclusion. Moreover, the Verification plug-in actually supports only two software applications.

Future works will be devoted to better evaluate the feasibility of our approach, as well as to consolidate and extend the developed software. We also plan to test our tools with a real e-learning by doing case, in order to understand the limits and the real advantages of our approach. In the medium term, we plan to build a repository of possible exercises supporting our approach, to the end of providing a reasonable experimental support for our tools.

Acknowledgements

Part of this work has been the subject of master thesis works tutored by the authors. A big thank then to Pamela Zapparoli, Vincenzo Rallo, Elisa Silenzi, and Giampaolo Paderno.

References

M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P.Torroni. (2008) Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM Transaction on Computational Logic (ToCL), Vol. 9, No. 4, pp. XX-YY. 2008.

M. Baldoni, C. Baroglio, B. Demo, V. Patti, and L.Torasso. (2003) E-learning by doing, an approach based on techniques for reasoning about actions. In G. Adorni, L. Sarti, and G. Vercelli, editors, *Proc. of 2nd Workshop on Artificial Intelligence & E-Learning*, pages 16-24, Pisa, Italy, September 2003.

J. Jaffar and M.J. Maher. (1994). Constraint logic programming: a survey. *Journal of Logic Programming*, vol. 19-20:503–582.

J.W. Lloyd. (1987). *Foundations of Logic Programming*. Springer-Verlag, 2nd extended edition.