Abstract

Nowadays multi-agents systems have been extensively used for the construction of intelligent tutoring systems, due to the high abstraction level. The MATHEMA model consists of an architecture to design of multi-agent systems based learning environments. In this paper we present the modelling and analysis of the MATHEMA using Coloured Petri Nets, scenarios, message sequence charts, temporal logic and model checking in a integrated way.

Keywords: Formal methods, Coloured Petri Nets, Multi-Agent Systems, Formal Software Development

1 Introduction

The MATHEMA [3] is an interactive learning environment model, based on multi-agent systems, for mathematical problem solving. The MATHEMA environment is still under development and among other components, the major one is a society of intelligent tutoring agents that interact through an interface with a given learner. These agents interact in a cooperative fashion in order to promote the teaching/learning activities in an interactive computer based learning environment. This paper presents the modeling, analysis and verification of the MATHEMA model by means of Colored Petri Nets [6].

Petri nets is a formal method with a graphical representation used to model concurrent systems. In the context of this work we consider Hierarchical Coloured Petri Nets (HCPN) as the modelling language [6]. It incorporates data types and hierarchy concepts to ease the modelling task. Coloured Petri Nets have been extensively used in many different applications [7], and the modelling activities are supported by a set of computational tools named Design/CPN [5], that provides a graphical environment for editing, syntax verification, as well as behavior analysis and verification of Colored Petri Nets models.

2 The MATHEMA

The MATHEMA architecture was defined aiming at the development of multi-agent intelligent tutoring systems based on problem solving cooperative based learning. Therefore, learning activities is based on problem resolution, instruction, hints, and explanations. In Figure 1 the high-level architecture for MATHEMA is shown.

The remaining of this paper is organized as follows: in Section 2 we introduce the MATHEMA model and in Section 3 the architecture and the functional scenarios for the MATHEMA model. In Sections 4 and 5 we introduce CPN basic concepts, the models, and the analysis respectively. Finally in Section 6 we present some conclusions.
3 Architecture and Scenarios

In this section the focus is the architecture for an agent in the SATA. The reader should refer to Figure 2 in the remaining of this section. The tutor system (TS) implements the mechanisms to promote the cooperative interactions between an agent and the learner during the learning process. Indeed, the learner is the intelligent entity and is divided in the following components: mediator, reasoners, and resources base. In what follows we conceptually define these components.

The mediator implements the interaction mechanisms with the Interface Agent and thus with the learner. It also selects the adequate reasoner in order to solve the task defined by the learner. Reasoners are the components that implement the pedagogic functionalities of the intelligent tutoring system. The reasoners are organized as container modules that compose the tutor system of an agent defined for the SATA, as are: tutor module, expert module and the learner modelling module. These three modules are explained in what follows.

The tutor module implements the pedagogical interactions with the learner. Its main functionality is, based on the reasoners, to select the resources that will be made available to the learner, namely, pedagogical tasks manager. Besides this, the tutor module must reply to the questions of the learners, problem solver, evaluate the answers given by the users, evaluator, and to define the next action that will be taken the system in order to improve the performance of the learner based on a cognitive diagnosis, remediator.

The expert module implements the subjective problem evaluation functionalities for the learner. Its response is inferred based on production rules stored in a knowledge base. This functionality is provided by the reasoner inference engine.

The learner modelling module is responsible to acquire, maintain, and represent the individual information about the learners. Such information are the used to define the best teaching strategy to be applied for a given learner. The functionalities of this module, implemented by its reasoners, are: (i) the organization and storing of all pedagogical knowledge already shown to the learner, history manager; (ii) the learning level for each learner for a given pedagogical unit, a domain partition, the profile manager; (iii) the definition of the cognitive level of each learner according to the profile, the cognitive diagnostic.

The resources base is the module that makes available the pedagogical resources, production rules and models for learners. The access to this module and its repositories are made through a resource access manager reasoner. The repositories store: (i) the resources related to the pedagogical knowledge with the definitions, examples, exercises, and hints; (ii) the information related to the profile of the learner and the resources already studied, namely, learner model; inference rules used by the inference machine to evaluate the answers given by the learner, namely, knowledge bases.

3.1 Functional View of an Agent

In Figure 2, it is shown both the functional view for the tutor system for an agent in the SATA taking into account the interactions with the other systems, such as the interface agent, and through it, with the learner. The functional scenario starts when a learner asks for the multi-agent intelligent tutoring system for a resource, as said before it can be a definition, an example, an exercise, or a hint, the solution of a proposed problem or the evaluation of a answer for a problem proposed to learner by the system. Based on this initial state for the scenario in what follows we discuss the sequence of events and interactions that will take place.

1. The learner asks the interface agent for a resource of a pedagogical unit.
2. The interface agent selects a agent in the SATA responsible for pedagogical unit and forwards the requisition to it. This one is the supervisor agent for the current actions.
3. The mediator of the agent receives the requisition and selects the reasoner to execute the requested task.
4. In the case that the learner asked for a problem resolution proposed by him, the reasoner then selects a problem solver (see the problem solver box labeled with A in Figure 2, that executes the following steps: (a) It divides the problem to be solved in smaller sub-problems. (b) Solves each one of the sub-problems. (c) In the case that there are sub-problems it cannot solve, it forwards them to the social system. The social system allocates other agents solve them. (d) When all the sub-problems are solved the answer is then returned to the mediator (see step 7).
5. If the learner asked for a resource, the selected reasoner is then the pedagogical tasks manager (label B in Figure 2), and then executes the following steps: (a) Asks the mediator (D) for the next type of resource to be presented to the learner according to his cognitive profile.
The mediator asks the profile manager (F) to recover the rank of the learner with respect to the pedagogic unit being studied.

The profile manager as the history manager (G) for the history of resources already given to the learner for the current pedagogical unit.

The history manager asks the resources access manager (I) for the list of resources already given by the learner and return it to the history manager, that then returns it to the profile manager.

Based on the learner history, the profile manager defines his quantitative knowledge level, based on the previous scores obtained for the other resources belonging to the current pedagogic unit. Then, it returns this score to the mediator.

The mediator gets the quantitative score and forwards it to the cognitive diagnostic (E) so that it can identify the qualitative level of the learner with respect to the current pedagogic unit, such as, for example, basic, intermediary, or advanced.

The cognitive diagnostic then returns the qualitative level of the learner to the mediator, that then defines the type of resource to be given to the learner. It forwards them, the level “n” and resource “r” to the pedagogical tasks manager.

The pedagogical tasks manager then asks for a resource “r” and level “n” to the resources access manager to then return the resource to the mediator (see step 7).

If the learner asked for an evaluation of his answer to a problem previously given by the system (what would occur in step 5) the selected reasoner is evaluator (C), that then executes the following steps:

(a) If the problem is an objective one, the evaluator verifies if the correct answer to the question is the same as the one given by the learner and returns the score to the mediator (step 7).

(b) In the case that the problem is subjective, it is forwarded to the inference engine (H) that divides it in smaller parts.

(c) The inference engine then asks for the resources access manager for knowledge inference rules and tries to validate each part of the solution.

(d) When the parts are validated, the whole solution is then validated. Therefore, the final score is returned to the evaluator.

(e) If some parts cannot be validated, they are forwarded to the social system, and then other agents are allocated to evaluate them.

(f) The evaluator then updates the historic of the learner and returns the score to the mediator (see step 7).

Based on the answer given to the learner, the mediator forwards it to the interface agent.

The interface agent then forwards it to the learner: the resolution of a problem proposed by the learner, a resource or the evaluation of a problem proposed by the system together with the score.

### 3.2 Social System

The social system is composed by the following modules:

(i) ask allocator: implements the allocation of the tasks defined by the tutor system; (ii) task allocator: defines the capabilities of the agents in the society; (iii) Coordinator: manages the cooperation between the agents during the resolution of the tasks, for example, when there are dependencies; (iv) cooperation: executes the cooperations, instantiating communication dialogs; (iv) dialogs: represent the instances of communication. In Figure 3 the interaction among these modules is illustrated and explained as follows: (i) the task allocator receives a set of tasks from the tutor system; (ii) the task allocator based on the social knowledge searches for agents that can solve them; (iii) the coordinator receives the set of tasks with the respective solvers defined; (iv) the coordinator manages the creation of cooperation processes; (v) each cooperation process starts a communication dialog with the agents in the SATA. The last system shown in Figure 2 is the distribution system that implements the functionalities for network connection. This system is discussed in this paper. The reader may refer to [8] for a detailed discussion.

### 4 Modelling the MATHEMA

In this paper we present the complete model for the MATHEMA. In this paper we modelled the tutor system, the social system, and integrated these models to the distribution system presented in [8]. The presentation of the modelling activity will follows the same order used to introduce the architecture of the MATHEMA. The models we build and analyzed using the set of tools named Design/CPN [4, 5].

In Figure 4 the hierarchy page for the model is shown. It has been graphically organized to be similar to the architecture diagram shown in Figure 2 and explained in Section 2. The model page for the mediator is shown in page Mediator, see Figure 5. A token in place Receive Task From AI models the situation when the model requests an action from the mediator module, please refer to the description in Section 2. Then transition Select Reasoners occurs. The function getReasoner(· · ·), not detailed in this paper, returns the reasoner that implements the functionalities needed to answer the request of the learner. As said in Section 2 these reasoners may be: PS for a Problem Solver, PTM for a Pedagogical Tasks Manager, or EV for an Evaluator. Therefore only one of the output transitions for place Ready to Select will occur, depending on
the selected reasoner, namely Select_PS, Select_PTM e Select_Me. Observe that transition Send_To_PS, Send_To_Ps, and Send_To_Pm, are substitution transitions, denoted by the rectangle with and HS next to these transitions. Please observe the Hierarchy page to identify these substitutions. After the execution of the models for each one of the reasoners a reply is sent back. For the problem solver module we have a token in place Problem_Solved. Then, transition PS_Return_To_IA occurs and a token is put on place result, modeling the end of the computation for the given reasoner. In the other two cases the behavior is similar and can be inferred from the model. Finally, transition Return_To_IA occurs and the answer given by the reasoner is put on place Return_Result_To_IA, and therefore to the InterfaceAgent page, since this is the output place.

5 Analysis Results

The use of formal modeling of systems allows the verify of a system before it is built or after built to identify design problems and improve a existing system. There are several ways to verify a system, such as simulation, model checking, among others. Specifically using the Design/CPN tool we use the simulation, Message Sequence Charts (MSC), and model checking. In this Section we discuss how we analyze the model using simulation, MSCs and model checking.

Based on simulations and defined scenarios for the behavior of the model we build an MSC to verify the interesting characteristics of the MATHEMA architecture, such as flow of the student interactions, and the decisions of the mediator based on the student requisitions. The Design/CPN tool has a library to allow automatic drawing of MSCs. For example, suppose that we want to see the simulation flow among the high level entities defined for the MATHEMA architecture, the student, the interface agent, the SATA, the social system, and the distribution system. In Figure 6 we can see an MSC for flow. Initially the student do a requisition that can be one of these: a solution of a subjective problem to be scored; a solution of an objective problem to be scored; a problem to be solved by the system; or the asking for a resource, like an explanation of a specific topic. The system receives the student requisition by the interface agent. Then the SATA executes it and decide to send or not to the social system the request based on the specific needs to produce an answer to the learner. For example, the problem solver needs some cooperation and distribution activities to be performed.

It is important to point out that simulation, a presented before, does not guarantee that the system always behaves as expected, since only one thread of execution is captured. Therefore we need to prove the behavior for all possibilities. To do this, we used temporal logic to describe the desired scenarios, then build the state space, and perform model checking to verify if the desired properties is satisfied for all execution threads. We used the ASK/CTL [1] library to perform model checking [2].

For the scenario described above, we need to prove that for all possible requests, the student always gets a valid reply. The formula below specify this property:

```plaintext
fun PA n = (Mark.Student'END 1 n = ((1,LEARNER_1)!!empty));
val formula = EV(NF("Final",PA));
```

The predicate PA, if true, specifies that there is a valid token at the place representing the situation where the learner gets a reply, in this case the place is named END and the page with this place is named Student, given by Mark.Student’END. Observe that we need to define this
boolean predicate in order to perform model checking over propositional formulas, such as the one given above. The interpretation of the formula, given the predicate PA, is that PA will be eventually true, see the temporal modality EV. The proof the this formula guarantees that several important characteristics of the system are correctly modelled, such as: the model always stops, in the case that it always possible to reach a state for which we have a token in place END in page Student; there are no deadlock situations; the model always produces a valid reply for a valid request; and all requests are treated by the system.

For another scenario, suppose now that we want to verify a specific part of the model, the mediator, as described above. This page is one of the most important because it models the mechanisms to decide which reasoner is selected on the learner request. We can do a step-by-step simulation to observe whether it behaves as expected. Then, after performing some simulations, and the system behaves as expected, we can perform the scenario verification in the same way as defined for the previous case. The formula below specify the correctness of the mediator page:

```plaintext
val formula = EV(EXIST_UNTIL(NF("Select",PA),
                         (NF("Measurer",PB))));
```

The predicate PA, if true, specifies that there is a valid token at the initial place of the mediator page, Ready_To_Select (please see Figure 5, and PB, if true, specifies that there is a valid response at the place modelling that a answer was returned, place Result, despite of the reasoner invoked by the mediator. For validation reasons we may need to verify the specific request treated by the mediator. The question to answer is: does the mediator always call the right entity based on the request? We can observe the processing of requests building MSCs for each possible case. First, we want to see the decision based on the request.
of a resource. In Figure 7 we can see the MSC for this case. The mediator receives a resource request and call the Pedagogical Task Manager, what a correct situation. Afterwards, it sends back the response to the mediator. Therefore we can conclude that our model behaves as expected. But it is always true? In order to answer this question, we defined formula below to prove that this kind of request is always processed in the right way.

\[
\begin{align*}
\text{val formula} =& \text{EV}(\text{AND}(\text{NF("Select",PA)}), \\
& \text{FORALL\_NEXT}(\text{NF("Evaluator",PB)}))
\end{align*}
\]

Predicate PA, if true, specifies that there is a token in the Ready\_To\_Select place for the Mediator page that contains a resource request, and predicate PB specifies that there is a token in the request path to pedagogical task manager entity, a token in place Ready\_To\_PTM. Thus, given PA and PB, the formula specifies that when PA is true then PB will be eventually true for any possible next state, where the next state is the immediate successor of the actual state.

The second possible request is related to an answer from the learner to a subjective or objective problem. As said before, this answer needs to be evaluated and scored by the system. This is performed by the evaluator entity. The formula below specify this reasoning.

\[
\begin{align*}
\text{val formula} = & \text{EV}(\text{AND}(\text{NF("Select",PA)}), \\
& \text{FORALL\_NEXT}(\text{NF("Evaluator",PB)}))
\end{align*}
\]

Predicate PA, if true, specifies that there is a token at the initial place for the mediator page with the request of a resource measurement, and PB, if true, specifies that there is a token in the path to the evaluator entity. Thus, the formula specifies that when PA is true, PB will be true for any possible next state. The reasoning here is similar to the previous case. The last possible request is a problem solver one. The request at the initial place for the mediator page with the request of an evaluator entity page. The formula specifies that there is a problem solver token in the path to the evaluator entity. Thus, the formula below to prove that this kind of request is always true? In order to answer this question, we defined formula below to prove that this request is always processed in the right way.

\[
\begin{align*}
\text{val formula} = & \text{EV}(\text{AND}(\text{NF("Select",PA)}), \\
& \text{FORALL\_NEXT}(\text{NF("Evaluator",PB)}))
\end{align*}
\]

Thus, predicate PA specifies that there is a problem solver request at the initial place of the mediator page, and PB specifies that there is a token in the path to the problem solver entity page. The formula specifies that when PA is true, PB will be true for any possible next state.

6 Conclusions

In this paper we presented the concepts of a multi-agent base interactive learning environment model named MATHEMA. This model is being developed and applied to different formal domains in the last year. The need to redesign and define a software framework to develop other kind of applications motivated us to develop a formal model for the MATHEMA model. Such formal model is being very helpful in this new design.

The focus of this paper was to demonstrate how formal methods, particularly Coloured Petri Nets can be effectively applied to help in the design of such complex system. We used a set of tools to edit, and perform analysis in the obtained model. The analysis were mainly guided by defining scenarios and verifying them, in a first moment, based on the automatic construction of message sequence charts. Since this is not enough to prove the correctness of the model for the given scenario we applied temporal logic to specify the scenarios and model checking to prove correctness.

We can conclude that the formal approach shown in this paper and applied for the MATHEMA models is very useful to help in complex designs.

We are currently building the software framework and the components in JAVA for the model described in this paper.

References


