One for All and All in One
A learner modelling server in a multi-agent platform

Isabel Machado[1], Alexandre Martins[2] and Ana Paiva[2]

[1] INESC, Rua Alves Redol 9, 1000 Lisboa, Portugal
[2] IST and INESC, Rua Alves Redol 9, 1000 Lisboa, Portugal
Emails: {Isabel.Machado, Alexandre.Martins, Ana.Paiva}@inesc.pt

Abstract. For the past few years several research teams have been developing intelligent learning environments (ILE) based on multi-agent architectures. For such type of architectures to be possible, the agents must have specific roles in the architecture and must be able to communicate in between them. To handle such needs, we have established a generic multi-agent architecture - the *Pedagogical Agents Communication Framework* (PACF). In PACF a set of agents were defined, their roles established, and their communication infrastructure built. Such communication infrastructure is based on a subset of the KQML language. There are two main general agents in PACF: the *Server* that acts both as a facilitator in the KQML sense and as an application-independent name server; and a Learner Modelling Server (LMS). The LMS can be used by several applications (composed by several agents) and each one can adapt the modelling strategy to its needs through the parameterisation of three configuration files: one that provides the application domain structure and the others the learner modelling strategies. Through this parameterisation the applications define how the LMS will model their learners. The LMS keeps one single database for all the learners being modelled by all the agents, allowing different applications to access to the same learner model simultaneously. These different applications can share parts of the learner models provided that they use the same domain ontology in the modelling process. This architecture has been used in a Web based distance learning scenario with two different ILEs.

1 Introduction

If we want to develop Intelligent Learning Environments (ILE) that aim to adapt the learning situations to the learners' state of knowledge, we have to maintain learner a model. Such models can be more or less complex, and can be used for either deep adaptation to the learners' preferences, state of knowledge etc., or for simple adaptation explicit in the immediate feedback provided by the environment. In both cases, the learner model itself needs to be acquired, maintained and updated along the learner interaction with the environment.

However, building such models is often left as a last job to be done in an ILE and very often remains undone. The main reason, in most cases, is that the cost of such adaptation is too high to pay the effectiveness achieved. Such high cost is due to three main reasons: first, the techniques for creating and maintaining these models are not at the finger tips of the application developers; second, very often such techniques rely on data that is difficult to acquire about the population of learners; and third even when learner models are acquired by one learning environment those models and techniques cannot be shared easily with other learning environments. In order to overcome the first problem some learner modelling shells were built so that the construction of learner modelling modules is simplified. Cases of such systems are BGP-MS (Kobsa, 1995) UM (Kay, 1995) and TAGUS (Paiva, 95). However, both systems do not allow an easy sharing of models nor they are Web oriented.

Indeed, as Web based learning environments become increasingly popular, individualisation has also became a need in such Web environments. Such individualisation has already been achieved through the use of learner modelling modules specific of some systems (see, e.g., Paiva and Machado, 1998). However, in order to use UM shells in Web scenarios we have to foster the independence between the applications and the learner modelling systems (associated with the shells). Such independence is better achieved through a client-server architecture (that is precisely the type of architectures we have on the Web). But that approach, on its own, does not guarantee us an easy online reuse of the learner models. To deal with such requirement, we need a standard communication mechanism between the server and the applications. So, in this paper we argue that such independence is better achieved with the use of a multi-agent architecture.

For the past few years, new types of architectures have been suggested for the construction of intelligent learning environments (ILEs), in particular multi-agent architectures (see, e.g., Frasson, 1996). However, one of the major problems that software developers face when following that approach is determining how the agents in the ILEs can communicate. This problem has already been subject to recent studies, in particular Ritter's protocol of communication (Ritter, 1997). However, the use of specific protocols prevents the inter-operability between agents developed by different organisations. Despite the lack of internationally accepted standards in the agent's domain, there are many situations in which a number of *de facto* standards can be used (Wooldridge, 1998). In this paper we will present a learner modelling server that is part of the PACF platform (*Pedagogical Agents Communication Framework* discussed in Martins et al., 1998). PACF adopted one of those *de facto* standards for establishing the communication between the agents - the KQML language (similar to the scenario presented by Paiva, 1996).

LMS is a learner modelling server, written in *Java*, that can be used by any application to acquire, store and maintain learner models. The application developer needs to give some configuration files to the LMS, which will take these files to perform the acquisition and maintenance of the models. The LMS, keeps the models of learners in a database, and updates these models according to the information given by the application and the configuration files provided initially. Indeed, the models are kept in a database, and then they can be shared by several applications (providing only, that they share the same ontology). The idea is that once agreed upon a common ontology, two applications can share their learner models, stored and maintained by the LMS.

This paper is organised as follows: first we will present the LMS within the PACF platform. Then we will describe in more detail the architecture and its modules, the domain and learner model composition and its parameterisation. The use of ontologies in LMS is focused and illustrated with a concrete scenario of LMS utilisation. Finally we will draw some conclusions.

2 LMS as a modelling agent in Multi-agent Platform

The LMS plays the role of a general agent in the *Pedagogical Agents Communication Framework (PACF).* The *PACF* describes how several agents implemented as components in a multi-agent system, can interact with each other to achieve the behaviour of an Intelligent Learning Environment (see Figure 1).

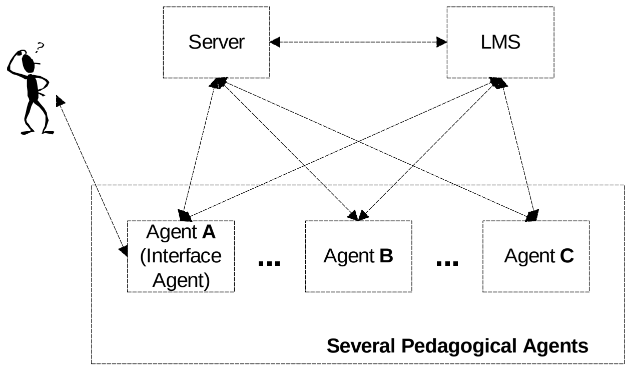

Figure 1 - The architecture of PACF

In this framework there is a set of agents that play different roles in the multi-agent learning environment. These agents can be, for example, interface agents that interact with the learner, or tutor agents that are expert in the domain. The agents must communicate exchanging messages that are called *performatives*, according to the KQML language (see Finin (1993, 1995)).

The KQML language can be understood as having three levels (Thirunavukkarasu, 1995): the *content level*, the *communication level* and the *message level*. The *content level* specifies the actual content of the message conveyed in the application's representation language, but it is in the *message level* that the actual message is built. In the framework we consider that the agents may present different capabilities and/or responsibilities in relation to the others, therefore, in order for agents to identify each other' roles, we provided a very simple agent taxonomy which gives an indication on the overall behaviour of a given agent type. Such taxonomy distinguishes two types of agents: general agents and application specific agents.

A general agent plays the same role in several different scenarios and offers the same set of functionalities in all situations. The *PACF* defines two of such general agents: the *Server* and the LMS. The *Server* acts both as a facilitator in the KQML sense and as an application-independent name server. The Server's address is known at system start-up time and the first action that all other agents perform is to register their own address and taxonomic type to that server using a KQML advertise *performative.* The server maintains a simple agent database, which is used to get agents mutually acquainted when they request to indicate another agent that is able to perform a certain task. The server supports multiple ontologies by keeping the ontology information along with each agent that complies with that ontology. This issue will be important in the sharing of the learner models.

The application specific agents are defined by each client application and their roles are specific within that application's domain. For example, an application agent that is usually found in most of the ILEs would be a domain expert agent. Another would be a tutoring agent, which task would be to interpret the data contained in the learner models (kept in the LMS) and provide adequate learning situations to the learner.

3 LMS

Within the ACF architecture the LMS plays the role of a learner modelling server. Learner modelling is the key to an individualised learning. Our main research goal was to develop a learner modelling server which could be used concurrently by multiple applications and that could also provide a reliable way of sharing the learner models between a set o applications.

3.1 The architecture of the Learner Modelling Server

With this aim in mind we based the LMS architecture on a client-server model (see Figure 2), which relies on *Java Remote Method Invocation (RMI)* protocol. With this architecture most of the learner models' manipulation is handled by the LMS, and only query results are sent back to the client applications. Although, the architecture relies on the *Java RMI* protocol, the communication between the LMS and the client applications is established by means of a subset of the KQML language. The overall communication between the LMS and the client applications is performed in the following manner:

– the client applications send a KQML *performative* to the LMS, for example, to inquire for some element of a learner model, or to require an update on a particular learner model, etc.;
– the LMS interprets the received message, performs the required actions and replies with a *performative*, which content can be the questioned value or an acknowledgement concerning a performed task.

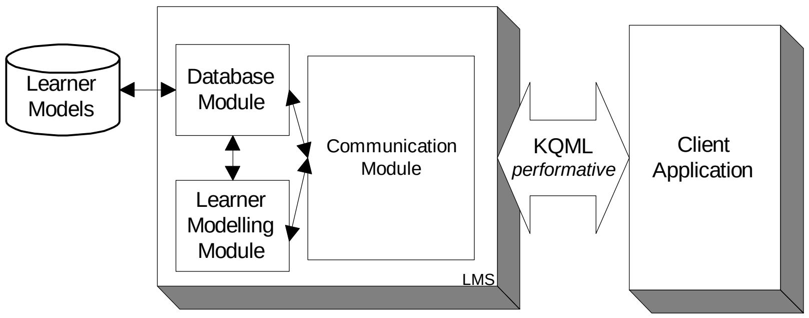

Figure 2 - The LMS architecture

To support these functionalities, the LMS architecture is modular and combines the following modules:

– the communication module - allows the communication with other agents through the use of a set of KQML *performatives*. Such performatives allow the manipulation and management of the learner models and also the establishment of the communication within the *PACF*;
– the database module – implements all the functions necessary to insert, update, delete, and select the learner models;
– the learner modelling module – executes the learner modelling cycle. It uses the parameterised learner modelling strategies to make decisions on the update of the learner models maintained by the server.

3.2 The generic Domain Model

In order to establish a basic domain model that can be used by the distinct client applications some restrictions were made:
– since the learner model aim to be used to store the learners state of knowledge with respect to a certain domain, then, we assumed that such domain is structured in *Topics*;
– while interacting with the application the learner performs *Actions* and those actions are indeed the observable behaviour of the learner that is used to infer his or her state of knowledge, preferences, etc.

With this simple structure we establish a baseline which allows the management of different domains associated to different applications, without any other different mechanism. Besides, improving the management of the domain models this generic structure makes easier the learner model sharing between the applications.

3.3 The Learner Models

For each client application the LMS stores the following information in the learner model:

– The declarative Knowledge Level associated with a particular *Topic*: $KL_{declarative}(Topic)$
– Information about some actions performed on the current *Topic*:
  ▪ the Knowledge Level associated with each *Action* that composes the current *Topic*: KL(Action);
  ▪ the number of contacts experienced by the learner with that *Action*.

This information is stored in the database and can represent the learners' beliefs about the domain or the learners' preferences or some learners' characteristics. The content of the model, differently from the TAGUS system, (Paiva, 1995), is thus defined entirely by the application developer, using the above types of structures.

3.4 The Parameterisation of the LMS

Each client application must specify how the LMS should model its learners. This specification is done by the definition of three configuration files: a domain structure file and two learning strategy files.

The domain structure file includes the definition of all the *Topics* that are introduced by the application and all the *Actions* that can be performed by a learner in a particular *Topic*.

Besides specifying the domain structure, the application must also define a weight value associated with each *Action*. This value translates the importance of that particular *Action* within the knowledge of a *Topic*. The LMS provides a graphical environment for the definition of domain structure and for the definition of the application specific parameters (see Figure 3).

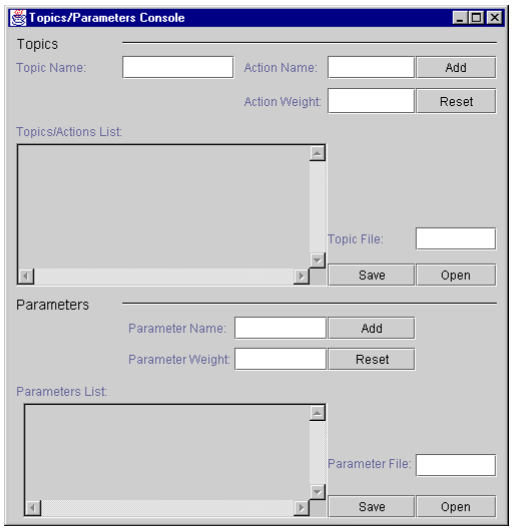

Figure 3 - Graphical environment for the definition of the domain structure and application parameters

The two learning strategy files include:

– the definition of application specific parameters, and;
– the definition of the classification and updating rules of the learner models.

The client applications have the possibility to define specific application parameters. These parameters provide the application with particular information about its learning process. Therefore, these parameters only have a meaning in that particular application. The application must define a set of rules that will assign and update values to those parameters and when defining them the application developer must also provide a default value for each of them.

The learning strategies adopted by the applications are defined through a set of rules. There are two kinds of rules: the *classification rules* and the *updating rules*. The classification rules define which learner behaviours should be included in a learner model. The syntax of the classification rules is the following:

CLASSIFICATION = $value_1$

$expression_{11}$ ... $expression_{1n}$

The expressions are relational expressions (using the usual relational operators: >, <, >=, <=, <> and =).

The semantic of each classification rule is expressed by the following condition:

```
if (∃i:compatible(expression_i)) and (¬∃j:incompatible(expression_j))
then
  CLASSIFICATION = value
```

To require an update of the learner model the client application must identify the current *Topic* and *Action*, and also define which are the correspondent updating expressions that should be used to characterise that particular learning situation. These expressions will have to match some (one) of the preconditions of a classification rule. Consequently, three distinct situations can arise, one of the preconditions of a classification rule may be:

– compatible - when it matches with one of the updating expressions and the precondition is verified;
– incompatible - when one of the updating rules invalidates the precondition, i.e., matches with one of the updating expression but is not verified;
– indifferent – when none of the updating rules matches any of the preconditions.

Therefore, one rule can be applied, in order to accomplish a request for a learner model update, if at least one of the preconditions is compatible and there is not any incompatible updating expression. This way, it is possible that there can exist several classification rules that could be applied in that particular updating request. In this case, the "winning" rule is the one that has a higher number of matches with the updating rules and if several classification rules have the same number of matches the LMS originates an internal message and no update is performed.

After finding which is the most suitable classification rule for that updating request, we have to define the set of actions that will constitute the updating rule. The updating rules will be applied to change the content of the learner models. The syntax of a updating rule is as follows:

```
CLASSIFICATION = value_1
update_11
```

The updating expressions assign values to variables (`variable = aritExpression`), and the `aritExpression` can contain the usual arithmetic operators: `+`, `-`, `*` and `/`. The semantic associated with an updating rule is:

if CLASSIFICATION = *value* then $update_1$ ...$update_n$

The definition of both kinds of rules can be done through the use of a graphical environment (see Figure 4).

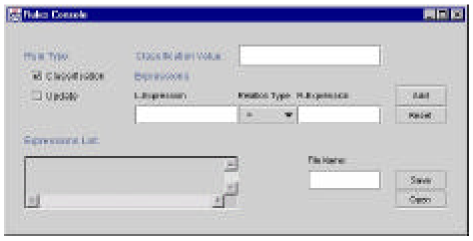

Figure 4 - Graphical environment for the definition of the classification and updating rules

The LMS configuration files are saved through the use of *Java* serializable mechanism. Through this mechanism is possible to store, in a file, *Java* objects without loosing any information associated with them.

3.5 The use of ontologies in the LMS

The learner models in LMS are maintained in a database that is accessible for any client application provided that it knows the password associated with a particular learner. So, the models stored by the LMS can be transparently shared amongst applications without the need for explicit communication between the applications, as in the case of PAT Online and Interbook (Brusilovsky, 1997). This sharing is nevertheless conditioned by the applications ontology type. The idea is that the client applications can negotiate and share the same ontology. As well as sharing the same ontology the applications need to also to share the knowledge about both domain structures, because they have to know which parts are common and can be used in the other learning strategy.

Therefore, in order for LMS to be a fully interoperable agent the learner models are structured according to the chosen domain ontology. This approach allows the agents from the different applications to interact with LMS based on a common grounded understanding of the elements that they have to specify to parameterise and use the LMS.

So far, in ILEs the use of ontologies has been centred on the description of learner models (see Mizoguchi, 1997). In here, such use is different in the sense that the ontologies are used for sharing the learner models, and thus they are domain ontologies.

4 Concrete Scenario

To illustrate this use we considered a concrete scenario in which there are two different web-based ILEs based on the PACF architecture. Both applications (ILEs) use the same LMS and interact with the general *Server*. The *Server* role is to establish the communication channels between all the agents in the platform (by indicating the remote locations of the existent agents). Both ILEs aim to teach the learners in the mathematics domain: one presents the linear functions domain and the other the quadratic functions domain. Hence, they have the same ontology type (*Mathematics*). The learner *Gil* with the password *iloveMaths* interacts with both ILEs. In the PACF platform there is an external service, which registers *Gil* within the PACF platform, and after this registration *Gil* is able to use any application on the this platform. In Table 1 and Table 2 we can see the domain structure of the two ILEs.

Linear Functions Application - Linear Functions Domain	
Topics	Actions
LinearF - y = mx + b	Identifying the functions zeros Graphical representation of the function Analytical representation of the function

Table 1 – Linear Functions Domain

Quadratic Functions Application – Quadratic Functions Domain

Topics	Actions
QuadraticF - $y = ax^2 + bx + c$	Reviewing the linear functions Introduction to the quadratic functions …

Table 2 – Quadratic Functions Domain

Gil starts to interact with the Linear Functions Application, solving correctly an exercise about the identification of the zeros of a function. Given that information the tutor agent of this application (TutorLF) asks the LMS to update of *Gil*'s learner model. Since the communication between all agents is done through the KQML language the update request would be done through the use of a KQML *performative*. For example,

```
(tell
  :sender rmi://leeds.inesc.pt/TutorLF
  :receiver rmi://bangkok.inesc.pt/LMS
  :reply-with q1
  :language JAVA
  :content (updateTopicAction(Mathematics Gil LinearF
         identifyZeros [(Content_Type=Exercise),(Success=Correct)] ?x)))
```

Gil carries on interacting with the two applications and reaches the end of his learning process in the Linear Functions application, continuing on the Quadratic Functions topic. When the tutor agent of such application (TutorQF) detects the presence of a new learner, he will ask the LMS for information about that new learner, sending the following *performative* to LMS:

```
(ask-if
  :sender rmi://athens.inesc.pt/TutorQF
  :receiver rmi://bangkok.inesc.pt/LMS
  :reply-with q2
  :language JAVA
  :content (infoTopic(Mathematics Gil LinearF ?x)))
```

Before presenting any contents to *Gil* the TutorQF inquires the LMS about *Gil*'s knowledge level. Since the TutorQF is only interested about the *Gil's* knowledge within the *Mathematics* ontology and since it also knows the existence of another application on this ontology (and with a common domain part), it will send a performative enquiring about *Gil* state of knowledge on this particular ontology. Considering that *Gil* had already achieved a good knowledge level on the linear functions domain (both applications need to understand what means that knowledge level) the TutorQF decides not to present to *Gil* the linear functions review but the quadratic functions introduction. If *Gil* achieves a relevant performance in that particular topic the TutorQF can update the knowledge level by simply sending the following *performative* to the LMS:

```
(tell
  :sender rmi://athens.inesc.pt/TutorQF
  :receiver rmi://bangkok.inesc.pt/LMS
  :reply-with q1
  :language JAVA
  :content (updateTopicAction(Mathematics Gil QuadraticF introduction
                [(Content_Type=Explanation),(Experience=FirstTime)] ?x)))
```

With this concrete scenario we have shown some of the advantages of assigning ontologies to our domain applications, specially the immediate advantages on the sharing the learner models between those applications.

5 Conclusions

In this paper we have introduced a new approach for a learner modelling systems embedded in multi-agent architectures. This approach allows not only the reusability of the software components (learner modelling processes) but also the sharing of the learner models. The main feature of this approach is the communication between the agents using a subset of KQML (a *de facto* standard) and the use of ontologies for the sharing of the models.

To illustrate our approach we have presented a learner modelling server for Web based learning environments that is an agent in the PACF framework. Using LMS in that multi-agent framework we have presented a scenario that shows how it works with two different applications sharing the same learner model.

References

Brusilovsky, P., Ritter, S. & Schwarz, E., "Distributed intelligent tutoring on the Web" in *Proceedings of AI-ED'97, 8th World Conference of Artificial Intelligence in Education*, Ed. B. du Boulay and R. Mizoguchi, IOS Press, 1997.

Finin, T., Weber, J., Wiederhold, G. & Genesereth, M., *Draft Specification of the KQML Agent Communication Language*, University of Maryland Baltimore County, 1993.

Finin T., Labrou Y. & Mayfield, J., "KQML as an agent communication language" in Software Agents, Ed. J. Bradshaw MIT Press, 1995.

Frasson, C., Mengelle, T., Aimeur, E. & Gouardères, G., "An Actor-based Architecture for Intelligent Tutoring Systems" in *Proceedings of Intelligent Tutoring Systems 96 Conference*, Lecture Notes in Computer Science, Springer Verlag, 1996.

Kay, J., "The UM Toolkit for Cooperative User Modeling". *User Modeling and User-Adapted Interaction* **4**(3), 149-196.

Kobsa, A. & Pohl, W., "The User Modeling Shell System BGP-MS". *User Modeling and User-Adapted Interaction* **4**(1), 59-106.

Paiva, A. & Self, J., "TAGUS – A User and Learner Modeling Workbench". *User Modeling and User-Adapted Interaction* **4**(3), 197-226.

Paiva, A. "Learner Modelling Agents" in *EuroAIED- Proceedings of the European Conference on Artificial Intelligence in Education*, Eds. P. Brna, A. Paiva & J. Self, Colibri, 1996.

Paiva, A., Machado, I., "Vincent, an autonomous pedagogical agent for on-the-job training" in *Intelligent Tutoring Systems 98 Conference,* Lecture Notes in Computer Science, Springer-Verlag, 1998.

Martins, A., Machado, I. & Paiva, A. "A KQML based communication framework for multi-agent ILEs" in the Pedagogical Agents Workshop of Intelligent Tutoring Systems 98 Conference, 1998.

Mizoguchi, R., Ikeda, M. & Sinitsa, K., "Roles of Shared Ontology in AI-ED Research" in *Proceedings of AI-ED'97, 8th World Conference of Artificial Intelligence in Education*, Ed. B. du Boulay and R. Mizoguchi, IOS Press, 1997.

Ritter, S., "Communication, Cooperation and Competition among Multiple Tutor Agents" in *Proceedings of AI-ED'97, 8th World Conference of Artificial Intelligence in Education*, Ed. B. du Boulay and R. Mizoguchi, IOS Press, 1997.

Thirunavukkarasu, C., Finin, T. & Mayfield, J., “Secret Agents – A Security Architecture for the KQML Agent Communication Language” in *CIKM’95 Intelligent Information Agents Workshop*, 1995.

Wooldridge, M. & Jennings, N., “Pitfalls of Agent-Oriented Development” in *Proceedings of Autonomous Agents 98*, Eds. K. Sycara & M. Wooldridge, ACM Press, 1998.