Agent Academy: An integrated tool for developing multi-agent systems and embedding decision structures into agents

Pericles A. Mitkas, Dionisis Kehagias, Andreas L. Symeonidis, Ioannis N. Athanasiadis

Department of Electrical and Computer Engineering
Aristotle University of Thessaloniki,
54 124, Thessaloniki, Greece,
Tel.: +30-2310-996349, Fax: +30-2310-996398

`mitkas@eng.auth.gr, {diok, asymeon, ionathan}@ee.auth.gr`

Abstract. In this paper we present Agent Academy, a framework that enables software developers to quickly develop multi-agent applications, when prior historical data relevant to a desired rule-based behaviour are available. Agent Academy is implemented itself as a multi-agent system, that supports, in a single tool, the design of agent behaviours and reusable agent types, the definition of ontologies, and the instantiation of single agents or multi-agent communities. Once an agent has been designed within the framework, the agent developer can create a specific ontology that describes the historical data. In this way, agents become capable of having embedded rule-based reasoning. We call this procedure "agent training" and it is realized by the application of data mining and knowledge discovery techniques on the application-specific historical data. From this point of view, Agent Academy provides a tool for both creating multi-agent systems and embedding rule-based decision structures into one or more of the participating agents.

1 Introduction

As agent technology has fairly recently emerged as a new paradigm for software development [1], the lack of standards has not allowed the proliferation of software engineering tools for the quick deployment of agent-based applications. Nevertheless, agent developers have at their disposal a plethora of existing agent constructing tools and development environments, albeit with limited capabilities in terms of the level of abstraction in the design and development process. Among the set of these agent development tools a mainstream can now be more easily identified than in the past years. In this respect, a quite desirable effort for agent developers is the creation of a software product that combines all widely used mainstream technologies in one tool. In an effort to mitigate this deficiency, we introduce Agent Academy (AA) [2], an integrated development framework, for constructing multi-agent applications and embedding rule-based reasoning into agents.

A survey of available agent-development tools [1] shows that the majority are Java-based applications that aim at facilitating rapid implementation of agent-based systems. Most of them [3-6] provide mechanisms to manage and monitor message exchanges between agents, as well as interface support for creating and debugging multi-agent systems. Many systems follow the mainstream of agent interoperability and adhere to FIPA specifications [3, 4], while others remain non-compliant to FIPA standards [5]. The existing development frameworks do not support the use of any particular reasoning tool in conjunction with the development environment they provide, but they do not prevent the agent developers from using other existing tools or implementing their own agent reasoning. In contrast, AA provides both a high-level, GUI-based environment for the design and development of agent-based applications and a training facility that creates rule-based reasoning into agents. A survey of existing tools for creating rule-based reasoning for agents is given in [7].

Agent Academy is implemented as an open-source project[1] using the JADE [8] development framework. This was done mainly for two reasons: a) JADE infrastructure is one of the most popular among agent developers today and we believe that it will help AA to gain easily acceptance by a large audience, b) JADE ensures compliance to FIPA standards, as defined in [9, 10]. Our framework provides a GUI that enables the design of ontologies with Protégé-2000 [11], as well as single agents or multi-agent communities, using common drag-and-drop operations. This capability facilitates agent developers to immediately instantiate large multi-agent systems with specifically defined behaviours.

Apart from using AA to construct a multi-agent system, an agent-application developer can take advantage of AA's "training module" and enable reasoning capabilities into agents. The main functionality of the training procedure relies on the transfer of "business intelligence" from available customer data to the newly created agents. The knowledge is extracted by performing data mining (DM) on the customer data and is transformed into rule-based decision models. These models, represented in Predictive Modeling Markup Language (PMML) [12], can be embedded into the agent and form its "intelligence". Agent Academy provides a means for updating and maintenance of the extracted knowledge and applied techniques, thus enabling agent retraining, which may result in the improvement of the extracted decision models.

This paper is structured as follows. In section 2 we introduce the system architecture. Section 3 outlines the procedure for the development of an agent-based application, while, in section 4, a presentation of the application of data mining techniques for agent "training" or "re-training" is given. Finally, section 5 concludes the paper and outlines future work.

[1] Agent Academy is publicly available under the LGPL. See http://sourceforge.net/projects/agentacademy/ for more details.

2. System architecture

The main architecture of AA is depicted in Figure 1. The basic components that comprise AA are:

1. The **Agent Factory** (AF). This component provides the main GUI of the application. It is responsible for enabling agent developers manipulate all functionalities provided by the platform in order to determine ontologies, design behaviours, construct agents and instantiate multi-agent systems. This GUI also triggers the DM procedure for embedding rule-based reasoning into agents.
2. The **Agent Use Repository** (AUR) acts as a storage facility to ensure interoperability between all AA components. AUR hosts a database system, which is responsible for storing ontologies, behaviours, agent types and historical data to be mined. Its functionality is mainly implemented using RMI.
3. The **Agent Training Module** (ATM) provides the main functions needed to insert a decision model extracted from the DM procedure into newly created agents.
4. The **Data Mining Module** (DMM) implements all data mining algorithms executed by the DM procedure, and enables the extraction of new decision models, which are embedded into agents via ATM.

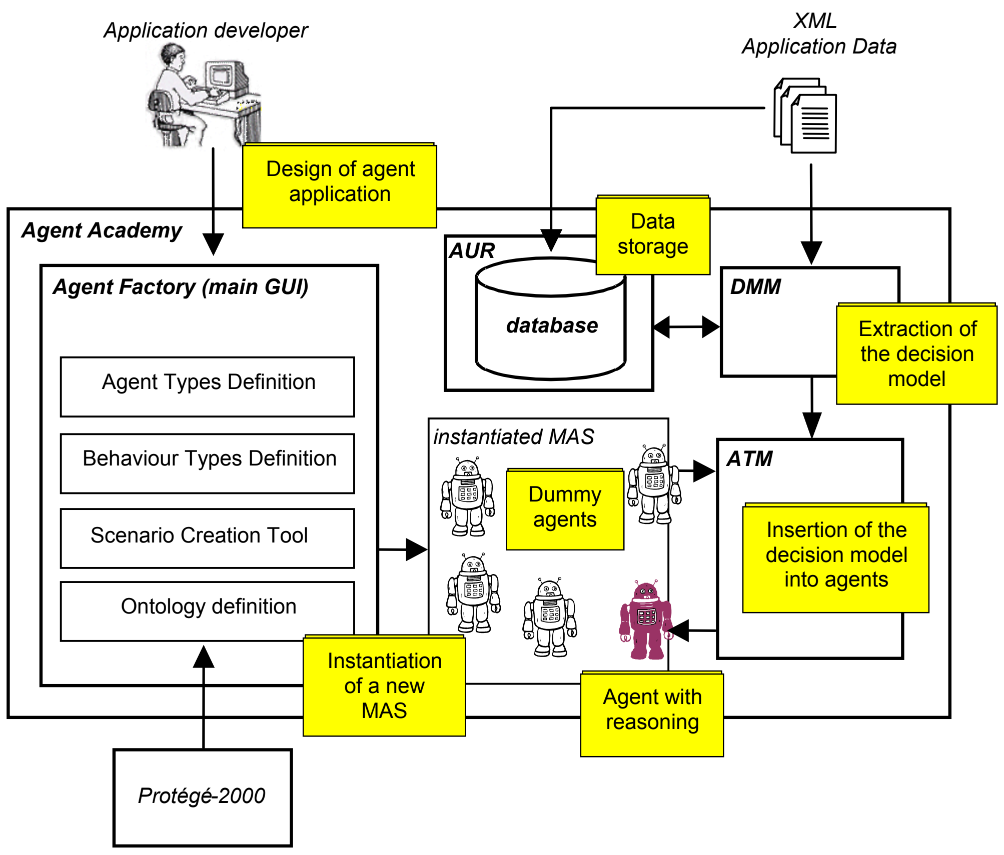


Fig. 1: Architecture of Agent Academy

The system architecture described above adheres to the procedure followed by agent application developers to develop a new multi-agent application. Hence, from a software-architecture point of view, a brief description of the main development procedure could consist of the following steps:

1. The agent developer initialises a new agent application (Agent Factory). For this purpose she/he determines all appropriate information about agents participating in the agent application, behaviour types, ontologies, etc.
2. All the above defined information is stored in a database for permanent access (Agent Use Repository).
3. The initially created agents possess no referencing capabilities ("dummy" agents). The developer may request from the system to create rule-based reasoning for one or more agents of the new MAS. These agents interact with the Agent-Training Module (Agent Training Module), which is responsible for inserting a specific decision model into them.
4. The latter is produced by performing DM on data entered into Agent Academy as XML documents or as datasets stored in a database (Data Mining Module). This task is performed by the DMM, another agent of AA, whose task is to read available data and extract decision models, expressed in PMML format.

All these steps are described in more detail in the following section.

3. Developing multi-agent applications

Through the main Agent Academy GUI, depicted in Figure 2, an application developer can design, instantiate and train one or more agents. The procedure for creating a new multi-agent system, which is referred to as a "scenario" in the context of AA, begins with the definition of the appropriate ontologies, which need to be shared by agents. This is done by the Ontology Design Tool (ODT). Next, the behaviours, which will be deployed are designed in the Behavior Type Design Tool (BTDT), shown in Figure 3. Based on these behavior templates, the AA user can assemble different reusable agent templates, using the Agent Type Definition Tool (ATDT). Finally, the Scenario Design Tool (SDT) enables the design and the instantiation of a MAS, by putting together several instances of the already defined agent types, and also specifying the interactions between them.

The appropriate actions for developing a new MAS are described in more detail in the remaining section.

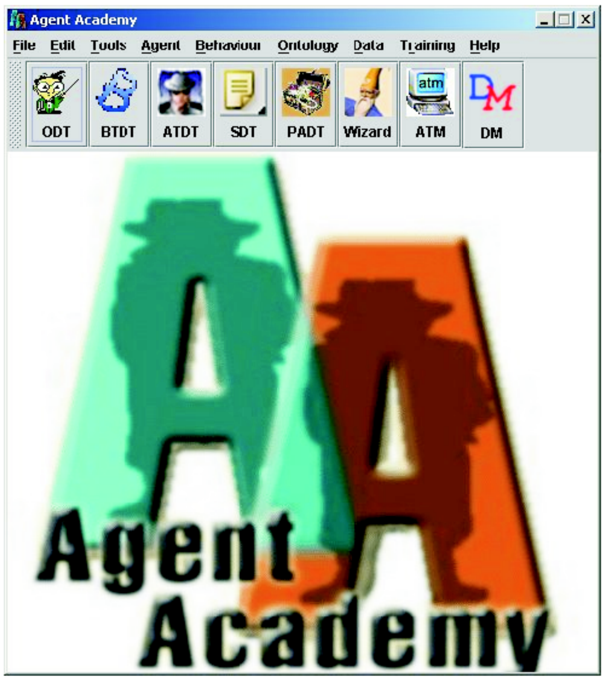


Fig. 2: Agent Academy's main GUI

3.1 Creating Agent Ontologies

A required process in the creation of a MAS, is the design of one or more ontologies, in order for the agents to interoperate adequately. The Agent Factory provides an Ontology Design Tool, which helps developers adopt ontologies defined with the Protégé-2000, a tool for designing ontologies. The RDF files

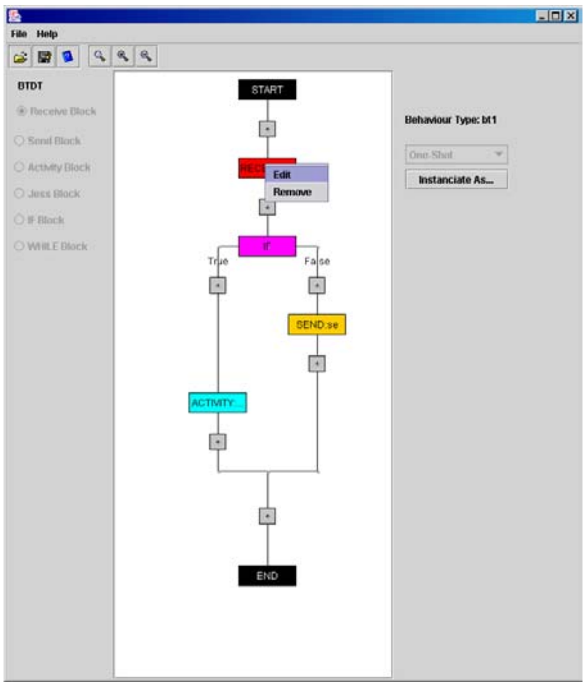


Fig. 3: Agent Academy's main GUI

that are created with Protégé are saved in the AA database for further use. Since AA employs JADE for agent development, ontologies need to be converted into special JADE ontology classes. For this purpose, our framework automatically compiles the RDF files into JADE ontology classes.

3.2 Creating Behaviour Types

The Behaviour Type Design Tool assists the developer in defining generic behaviour templates. Agent behaviours are modeled as workflows of basic building blocks, such as receiving/sending a message, executing an in-house application, and, if necessary, deriving decisions using inference engines. The data and control dependencies between these blocks are also handled. The behaviours can be modeled as cyclic or one-shot behaviours of the JADE platform. These behaviour types are generic templates that can be conFigured to behave in different ways; the structure of the flow is the only process defined, while the configurable parameters of the application inside the behaviour, as well as the contents of the messages can be specified using the MAS Creation Tool. It should be denoted that the behaviours are specialized according to the application domain.

The building blocks of the workflows, which are represented by nodes, can be of four types:

1. `Receive nodes`, which enable the agent to filter incoming FIPA-SL0 messages.
2. `Send nodes`, which enable the agent to compose and send FIPA-SL0 messages.
3. `Activity nodes`, which enable the developer to add predefined functions to the workflow of the behaviour, in order to permit the construction of multi-agent systems for existing distributed systems.
4. `Jess nodes`, which enable the agent to execute a particular reasoning engine, in order to deliberate about the way it will behave.

Fig. 3 illustrates the design of the behaviour for an agent that receives a message and, according to the content of the message, either executes a pre-specified function, or sends a message to another agent.

3.3 Creating Agent Types

After having defined certain behaviour types, the *Agent Type Definition Tool* is provided to create new agent types, in order for them to be used later in the MAS Creation Tool. An agent type is in fact an agent plus a set of behaviours assigned to it. New agent types can be constructed from scratch or by modifying existing ones. Agent types can be seen as templates for creating agent instances during the design of a MAS.

During the MAS instantiation phase, which is realized by the use of the MAS Creation Tool, several instances of already designed agent types will be instantiated, with different values for their parameters.

Each agent instance of the same agent type can deliver data from different data sources, communicate with different types of agents, and even execute different reasoning engines.

3.4 Deploying a Multi Agent System

The design of the behaviour and agent types is followed by the deployment of the MAS. The *MAS Creation Tool* enables the instantiation of all defined agents running in the system from the designed agent templates. The receivers and senders of the ACL messages are set in the behaviours of each agent. After all the parameters are defined, the agent instances can be initialized. Agent Factory creates *default AA Agents*, which have the ability to communicate with AF and ATM. Then, the AF sends to each agent the necessary ontologies, behaviours, and decision structures.

4. Using data mining algorithms for Agent Training

The initial effort for the implementation of such a development framework as the one presented in this paper, was motivated by the lack of an agent-oriented software-engineering tool coupled with AI aspects, as far as we know. The ability to incorporate background knowledge into an agent's decision–making process is arguably essential for effective performance in dynamic environments. However, agent-oriented software engineering methodologies deal with, both high-level, top-down iterative approaches and design methods for software systems [13]. Building a MAS with a large number of agents usually requires the reasoning to be distributed in many agents of the MAS community, reducing the degree of reasoning per agent. From our perspective, an agent-oriented development infrastructure should both provide high-level design capabilities and deal with the internals of an agent architecture, in order to be considered complete and generic.

For this reason, we have implemented, as a separate module of the overall agent-oriented development environment a mechanism for embedding rule-based reasoning capabilities into agents. This is realized through the ATM, which is responsible for embedding specific knowledge into agents. This knowledge is generated as the outcome of the application of DM techniques into available data. The other module, whose role is to exploit possible available datasets in order to extract decision models, is the DMM. Both ATM and DMM are implemented as JADE agents who act in close collaboration.

These two basic modules, as well as the flow of the agent training process are shown in Figure 4. At first, let us consider an available source of data formatted in XML. The DMM receives data from the XML document and executes certain DM algorithms (suitable for generating a decision model), determined by the agent-application developer. The output of the DM procedure is formatted as a PMML document. PMML is an XML-based language, which provides a rapid and efficient way for companies to define predictive models and share models between compliant vendors' applications. It allows users to develop models within one vendor's application, and use other vendors' applications to visualize, analyze, evaluate or otherwise use the models. The fact that PMML is a data mining standard defined by DMG (Data Mining Group) [12] provides the Agent Academy platform with versatility and compatibility to other major data mining software vendors, such as Oracle, SAS, SPSS and MineIt.

The PMML document represents a knowledge model that expresses the referencing mechanism of the agent we intend to train. The resulted decision model is translated, through the ATM, to a set of facts executed by a rule engine. The implementation of the rule engine is provided by JESS [14], a robust mechanism for executing rule-based reasoning. Finally, the execution of the rule engine becomes part of agent's behaviour.

As shown in Figure 4, an agent that can be trained through the provided infrastructure encapsulates two types of behaviours. The first is the basic initial behaviour predefined by the AF module. This may include a set of class instances that inherit the Behaviour class defined in JADE [10]. The initial behaviour is created at the agent generation phase, using the Behaviour Design Tool, as described in the previous section. This type of behaviour characterizes all agents designed by Agent Academy, even if the developer intends to equip them with rule-based reasoning capabilities. This essential type of behaviour includes the set of initial agent beliefs.

The second supported type of behaviour is the rule-based behaviour, which is optionally created, upon activation of the agent-training feature. This type of behaviour is dynamic and implements the decision model. In the remaining section, we present the details of the data mining procedure and we describe the mechanism for embedding decision-making capabilities into the newly trained agents.

4.1 Data Mining Techniques

The mechanism for extracting knowledge from available data, in order to provide agents with reasoning, is based on the application of DM techniques on background application-specific data [15]. From our experience with the application of our framework to an industrial scenario about supply chain management [16], we ascertained that the enterprise IT infrastructures generate and manipulate a large amount of data on a permanent basis, thus becoming suitable data providers that satisfy the purposes of DMM.

In the initial phase of the DM procedure, the developer launches the GUI-based wizard depicted in Figure 5.a and specifies the data source to be loaded and the agent decision attributes that will be represented as internal nodes of the extracted decision model. In Figure 5.b the developer selects the type of the DM technique from a set of available options.

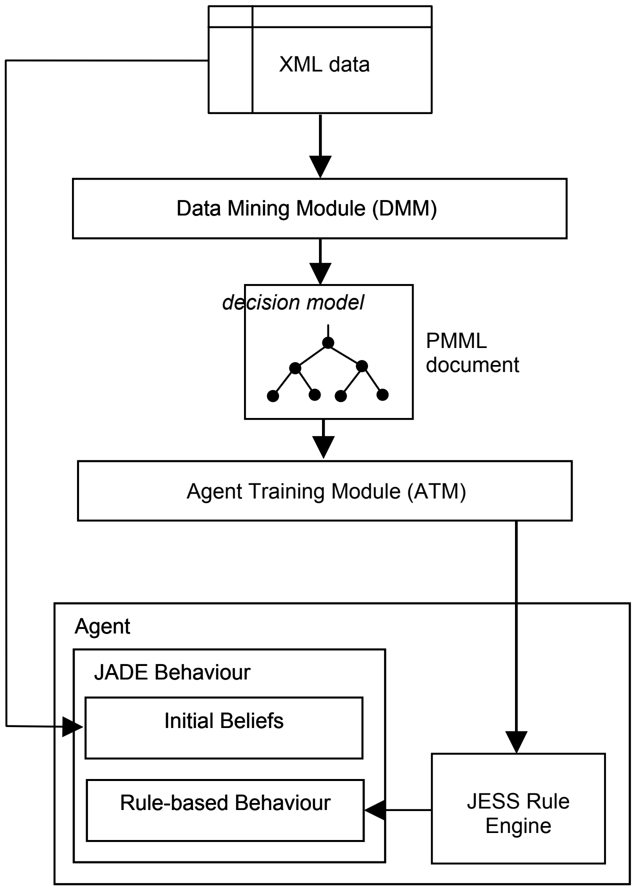
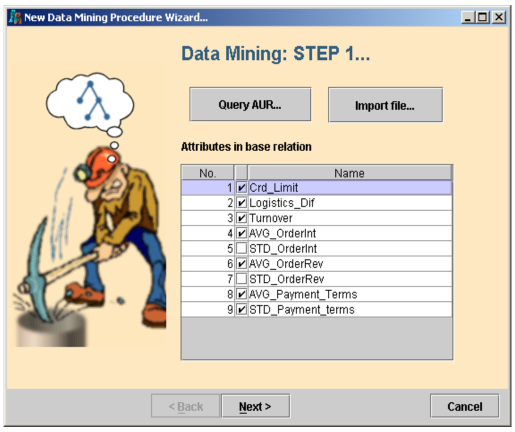


Fig.4 Diagram of the agent training procedure

Regarding the technical details of the DMM, we have developed the DM facility in our framework, by incorporating a set of DM methods based on the WEKA [17] library and tools and we further extended the WEKA API in order for it to support PMML (a later version of the WEKA API will have our extension included). Some other new DM techniques have also been developed but we will not mention them here, as this would be out of this paper's scope. For further information on the developed DM algorithms, please see [18].

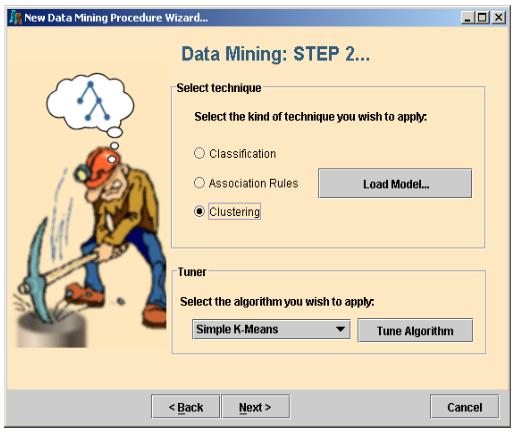
In Figure 6.a, we present an XML document, while the respective PMML output, which represents a cluster-based decision model, is shown in Figure 6.b. In order to generate the PMML output, we used the K-means algorithm to perform clustering on selected attributes described in the XML document. The dataset illustrated in Figure 6.a comes from the design of the formerly mentioned MAS [16] about supply chain management. The XML document concerns customer-related data. The PMML output shown in Figure 6.b, contains, apart from the extracted decision model, some other algorithm-specific details, such as the number of clusters produced, the attributes of the data set and the document version.



Attributes in base relation

No.		Name
1	✓	Crd_Limit
2	✓	Logistics_Dif
3	✓	Turnover
4	✓	AVG_OrderInt
5		STD_OrderInt
6	✓	AVG_OrderRev
7		STD_OrderRev
8	✓	AVG_Payment_Terms
9	✓	STD_Payment_terms

(a)



Select technique

Select the kind of technique you wish to apply:

( ) Classification
( ) Association Rules
(●) Clustering

Tuner

Select the algorithm you wish to apply: Simple K-Means

(b)

Fig.5 The two first steps of the DMM wizard

```
<Instances title="AUTH Data" author="Pericles
Mitkas">
    <Attributes>
        <Attribute>
                <Name>AVG_Order_Freq</Name>
                <Type>numeric</Type>
        </Attribute>
        <Attribute>
                <Name>AVG_Order_Rev</Name>
                <Type>numeric</Type>
        </Attribute>
        <Attribute>
                <Name>AVG_Payment_Trms</Name>
                <Type>numeric</Type>
        </Attribute>
        <Attribute>
                <Name>Crd_Limit</Name>
                <Type>numeric</Type>
        </Attribute>
        <Attribute>
                <Name>Logistics Dif</Name>
                <Type>numeric</Type>
        </Attribute>
        <Attribute>
                <Name>STD_Order_Freq</Name>
                <Type>numeric</Type>
        </Attribute>
        <Attribute>
                <Name>STD_Order_Rev</Name>
                <Type>numeric</Type>
        </Attribute>
        <Attribute>
                <Name>STD_Payment_Terms</Name>
                <Type>numeric</Type>
        </Attribute>
        <Attribute>
                <Name>Turnover</Name>
                <Type>numeric</Type>
        </Attribute>
    </Attributes>
    <Data>
        <Row>
              <Crd_Limit>-8.23599E-
2</Crd_Limit>
              <Logistics_Dif>-6.69967E-
2</Logistics_Dif>
              <Turnover>-0.138325</Turnover>
              <AVG_Order_Freq>-
0.64769</AVG_Order_Freq>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pmml_2_0.dtd>
<PMML>
  <Header copyright="CERTH" description="
Clustering Model of ERP data">
    <Application name="Agent Academy"
version="0.3" />
  </Header>
  <DataDictionary numberOfFields="9">
…
  </DataDictionary>
  <ClusteringModel modelName="ERP-
org.agentacademy.modules.dataminer.filters.Repla
ceMissingValuesFilter" modelClass="centerBased"
numberOfClusters="5">
    <MiningSchema>
         …
    </MiningSchema>
    <ClusteringField field="AB"
compareFunction="squaredEuclidean" />
    <ClusteringField field="CL"
compareFunction="squaredEuclidean" />
    <ClusteringField field="TO"
compareFunction="squaredEuclidean" />
    <ClusteringField field="AOP"
compareFunction="squaredEuclidean" />
    <ClusteringField field="STDAOP"
compareFunction="squaredEuclidean" />
    <ClusteringField field="AOI"
compareFunction="squaredEuclidean" />
    <ClusteringField field="STDAOI"
compareFunction="squaredEuclidean" />
    <ClusteringField field="APT"
compareFunction="squaredEuclidean" />
    <ClusteringField field="STDAPT"
compareFunction="squaredEuclidean" />
    <Cluster name="0">
      <Array n="9"> -1.825885 17.36695 18.23353
-0.55470 -0.47161 7.96735 14.47850 0.40154
0.92600</Array>
    </Cluster>
    <Cluster name="1">
      <Array n="9"> 0.06525-0.27008-0.12450-
0.64679-0.71163-0.35276-0.21816-1.32502-
0.49305</Array>
    </Cluster>
    <Cluster name="2">
      <Array n="9"> 0.03786 0.04243 0.04831 -
0.08052 -0.17864 0.16450 0.09465 0.63661 0.00631
</Array>
    </Cluster>
```

(a) (b)

Fig.6 XML input (a) and PMML output (b)

The completion of the training process requires the translation of the DM resulted decision model into an agent-understandable format. This is performed by the ATM, which receives the PMML output as an ACL message sent by the DMM, as soon as the DM procedure is completed, and activates the rule engine. Actually, the ATM converts the PMML document into JESS rules and communicates, via appropriate messages, with the “trainee” agent, in order to insert the new decision model into it. After the completion of this process, our framework automatically generates the new “trained” agent into the predefined MAS. The total configuration of the new agent is stored in the development framework, enabling future modifications of the training parameters, or even the retraining of the already “trained” agents.

5. Conclusions and Future Work

In this paper we have presented Agent Academy, a multi-agent development framework for constructing multi-agent systems, or single agents. We argued that the existing tools and infrastructures for agent development are especially focused on the provision of high-level design methodologies, leaving out the details of agents’ decision-making abilities. In contrast, our framework can provide both a GUI-based, high- level MAS authoring tool and a facility for extracting rule-based reasoning from available data and inserting it into agents. The produced knowledge is expressed as PMML formatted documents. We have presented the functional architecture of our framework; we shortly demonstrated an indicative scenario for deploying a MAS and, finally, we discussed the details of the agent “training” process.

Through our experience with Agent Academy, we are convinced that this development environment significantly reduces the programming effort for building agent applications, both in terms of time and code efficiency, especially for those MAS developers who use JADE. For instance, one MAS, that requires the writing of almost 6,000 lines of Java code, using JADE, requires less than one hour to be developed with Agent Academy. This test indicates that AA meets the requirement for making agent programs in a quicker and easier manner. On the other hand, our experiments with the DMM have shown that the completion of the decision model generated for agent reasoning is highly dependant on the amount of available data. In particular, a dataset of more than 10,000 records is adequate enough for producing high-confidence DM results, while datasets with fewer than 3,000 records have yielded non-consistent arbitrary output.

The AA framework is the result of a development effort, which begun two years ago. The first stable implementation of AA has been released on September 2003, as an open-source product [19]. Since Agent Academy is a large project it involves a lot of beta-testing, to ensure bug-free functionality. In this context, a significant effort for the evaluation of the platform has been planned and initialised by the project developers.

Our near future work involves the finalization of the integration process and documentation of AA, as well as the exhaustive testing of the platform, by implementing three large-scale applications in the domains of real-time notification, web-based applications, and supply-chain management.

Acknowledgements

Work presented in this paper is partially funded by the European Commission, under the IST initiative as a research and development project (contract number IST-2000-31050, "Agent Academy: A Data Mining Framework for Training Intelligent Agents"). Authors would like to thank all members of the Agent Academy consortium for their remarkable efforts in the development of such a large project.

References

1. Lind J.: Issues in Agent-Oriented Software Engineering. In: First International Workshop on Agent-Oriented Software Engineering (AOSE-2000), Limerick, Ireland (2000)
2. Agent Academy's official web-site: `http://agentacademy.iti.gr/`
3. Nwana, H., Ndumu, D., Lee, L., Collis, J.: ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. In: Applied Artifical Intelligence Journal, Vol 13 (1) (1999) 129-186
4. Suguri, H., Kodama, E., Miyazaki, M., Nunokawa, H., Noguchi, S.: Implementation of FIPA Ontology Service. In: Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents Montreal, Canada (2001)
5. Gutknecht, O., Ferber, J.: Madkit: A generic multi-agent platform. In: 4th International Conference on Autonomous Agents, Barcelona, Spain (2000)
6. Jeon, H., Petrie, C., Cutkosky, M.: JATLite: a Java agent infrastructure with message routing. In: IEEE Internet Computing 4 (2) (2000) 87-96
7. Rahimi, S., Cobb, M., Ali, D., Paprzycki, M.: An Analysis of Intelligence-Enhancing Techniques for Software Agents. In: Proceedings of the 5th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando (2001)
8. Bellifemine F., Poggi A., Rimassa G., Turci P.: An Object-Oriented Framework to realize Agent Systems. In: Proceedings of WOA 2000 Workshop, Parma, Italy (2000), 52-57
9. Foundation for Intelligent Physical Agents, the: FIPA Developer's Guide (2001) available at: `http://www.fipa.org/specs/fipa00021/`
10. Bellifemine F., Caire G., Trucco T., Rimassa G.: JADE Programmer's Guide. (2001) available at: `http://sharon.cselt.it/`
11. Noy, N.F., Sintek, M., Decker S., Crubezy, M., Fergerson, R.W., & Musen, M.A.: Creating Semantic Web Contents with Protégé-2000. In: IEEE Intelligent Systems 16 (2): (2001) 60-71
12. Data Mining Group, the: Predictive Model Markup Language Specifications (PMML), ver. 2.0 available at: `http://www.dmg.org`

13. Tveit, A.: A survey of Agent-Oriented Software Engineering. In: NTNU Computer Science Graduate Student Conference, Norwegian University of Science and Technology, Trondheim, Norway (2001)
14. Java Expert System Shell (JESS): `http://herzberg.ca.sandia.gov/jess/`
15. Symeonidis, A.L., Mitkas, P.A., Kehagias, D.: Mining patterns and rules for improving agent intelligence through an integrated multi-agent platform. In: 6th IASTED International Conference, Artificial Intelligence and Soft Computing ASC, Banff, Alberta, Canada (2002)
16. Symeonidis A.L, Kehagias D., Koumpis A., Vontas A.: Open Source Supply Chain. In: 10th International Conference on Concurrent Engineering (CE-2003), Workshop on intelligent agents and data mining: research and applications, Madeira, Portugal (2003)
17. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann publishers, San Francisco, CA (2000)
18. Athanasiadis, I.N., Kaburlasos, V.G., Mitkas, P.A., and Petridis, V.: Applying Machine Learning Techniques on Air Quality Data for Real-Time Decision Support. In: First International NAISO Symposium on Information Technologies in Environmental Engineering (ITEE'2003), Gdansk, Poland (2003) available at: `http://danae.ee.auth.gr/en/pdf/itee2003.pdf`)
19. Agent Academy on SourceForge (2003): `http://sourceforge.net/projects/agentacademy/`