A Framework for Constructing Multi-agent Applications and Training Intelligent Agents

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

Department of Electrical and Computer Engineering,
Aristotle University of Thessaloniki,
GR541 24 Thessaloniki, Greece,
Tel.: +30-2310-996349,
Fax: +30-2310-996398
`mitkas@eng.auth.gr`,
{`diok,asymeon,ionathan`}`@ee.auth.gr`

Abstract. As agent-oriented paradigm is reaching a significant level of acceptance by software developers, there is a lack of integrated high-level abstraction tools for the design and development of agent-based applications. In an effort to mitigate this deficiency, we introduce Agent Academy, an integrated development framework, 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. In addition to these characteristics, our framework goes deeper into agents, by implementing a mechanism for embedding rule-based reasoning into them. We call this procedure "agent training" and it is realized by the application of AI techniques for knowledge discovery on application-specific data, which may be available to the agent developer. In this respect, Agent Academy provides an easy-to-use facility that encourages the substitution of existing, traditionally developed applications by new ones, which follow the agent-orientation paradigm.

1 Introduction

In the last years, agent technology has impressively emerged as a new paradigm for software development [1], which is expected to gain even wider acceptance among the software developers. One important contribution to this effort could be the provision of such tools and development environments that will enable the deployment of agent-based applications quickly and easily. As opposed to this envisioned situation, the current landscape of agent constructing tools is characterized by a plethora of agent development environments, which provide limited capabilities in terms of the level of abstraction in the design and development process of agent-oriented applications. On the other hand, the scope of agent tools and technologies, which dominate the mainstream of development trends in this field, is now becoming clearer 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 order to fulfill this demand, we have developed Agent Academy (AA) [2], an integrated framework for constructing multi-agent applications and embedding rule-based reasoning into agents, at the design phase.

The framework presented in this paper is implemented upon the JADE [3] infrastructure, ensuring a relatively high degree of FIPA compatibility, as defined in [4,5]. AA is itself a multi-agent system, whose architecture is based on the GAIA methodology [6]. It provides an integrated GUI-based environment that enables the design of single agents or multi-agent communities, using common drag-and-drop operations. This capability of the AA development environment helps agent application developers to build a whole community of agents with chosen behaviour types and attributes in a few minutes. Using AA, an agent developer can easily go into the details of the designated behaviours of agents and precisely regulate communication properties of agents. These include the type and number of the agent communication language (ACL) messages exchanged between agents, the performatives and structure of messages, with respect to FIPA specifications [7,8,9], as well as the semantics, which can be defined by constructing ontologies with Protégé–2000 [10].

All of the aforementioned characteristics of our development environment have been viewed from an agent–oriented software engineering perspective, since they provide essential elements for the design and the construction of a multi-agent system with pre-specified attributes. In addition to that, there is the AI perspective that deals with the reasoning capabilities of agents. In this context, our system implements a "training module" that embeds essential rule-based reasoning into agents. This kind of reasoning is based on the application of data mining (DM) techniques on possible available datasets. This methodology developed within AA, results in the extraction of agent knowledge in the form of a decision model (e.g. a decision tree). The extracted knowledge is expressed in Predictive Modeling Markup Language (PMML) [11] documents and stored in a data repository, handled by our development framework. The applied data mining techniques are, by definition, updateable as new data come into the repository. Thus, it is easy to update the knowledge bases of agents, by performing agent "retraining". This capability can be especially exploited in environments with large amounts of periodically produced data. A characteristic example of such an environment is encountered in almost all enterprise IT infrastructures, the vast majority of which are implemented following traditional development paradigms. To this end, our presented infrastructure is envisioned as a convenient tool that will encourage the development of new agent-based applications over the existing traditional ones, by exploiting available data.

The paper is structured as follows. Section 2 briefly reviews related work. Section 3 describes the architecture of our framework and illustrates the development process and the use of tools provided for the construction of a multi-agent system. In section 4, a detailed presentation of the agent "training" mechanism is given. Finally, section 5 concludes the paper and outlines future work.

2 Existing Tools and Applications

The growth in interest and use for agent technology motivated the development of different frameworks and environment to support the implementation of multi-agent systems. Most of them are Java-based applications that aim at facilitating rapid implementation of agent-based applications, by providing mechanisms to manage and monitor message exchanges between agents, and interface support for creating and debugging multi-agent systems.

An advanced open-source tool-kit providing a library of software components and tools that enable the rapid design, development and deployment of agent systems is ZEUS [12]. Although this system is FIPA compliant, it does not support agent mobility, as opposed to AA. Another development environment [13] is implemented as a multi-agent system, in a similar manner as AA, but it does not satisfy the requirements for FIPA compliance.

As far as compliance to FIPA standards is concerned, there is a development framework [14] that meets the FIPA specifications about Agent Management and Agent Communication Language, among others, as well as AA does. Another tool [15] for creating agent systems uses FIPA-ACL for agent messages, but implements its own naming register service, ignoring the relative FIPA specifications.

All of the aforementioned development frameworks do not facilitate the use of any particular reasoning tools, 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 the developed agents. A survey of existing tools for creating rule-based reasoning for agents is given in [16].

3 The Development Framework

Our development framework acts as an integrated GUI-based environment that facilitates the design process of a MAS. It also supports the extraction of decision models from data and the insertion of these models into newly created agents. Developing an agent application using AA involves the following activities from the developer's side:

a. the creation of new agents with limited initial reasoning capabilities;
b. the addition of these agents into a new MAS;
c. the determination of existing, or the creation of new behaviour types for each agent;
d. the importation of ontology-files from Protégé–2000;
e. the determination of message recipients for each agent.

In case that an agent application developer intends to create a reasoning engine for one or more agents of the designed MAS, two more operations are required for each of those agents:

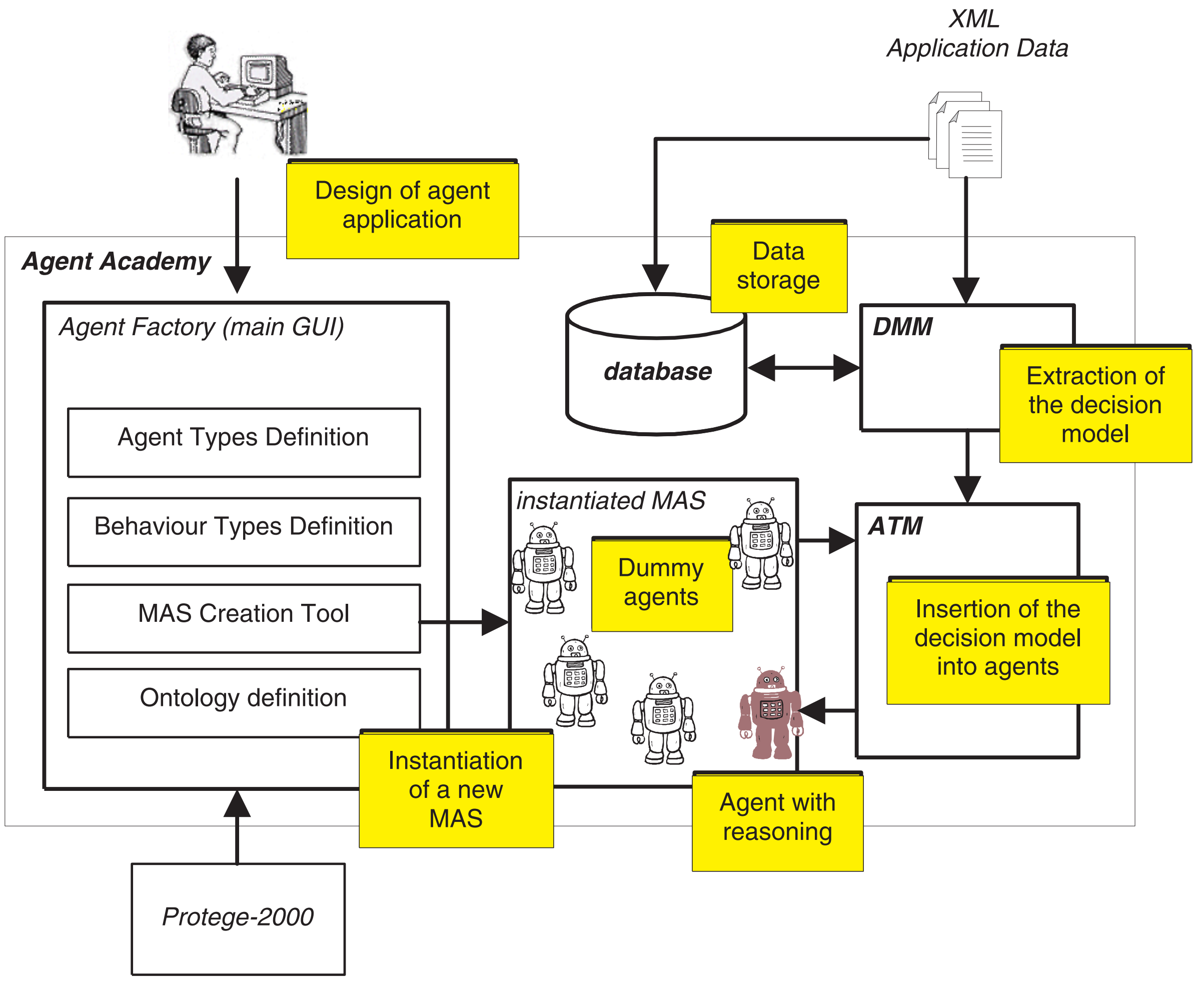


Fig. 1. Diagram of the Agent Academy development framework

– the determination of an available data source of *agent decision attributes*;
– the activation of the training procedure, by specifying the parameters of the training mechanism.

Figure 1 illustrates the Agent Academy main functional diagram, which represents the main components and the interactions between them. In the remaining section, we discuss the Agent Academy architecture, and we explain how the development process is realized through our framework.

3.1 Architecture

The main architecture of AA is also shown in Fig. 1. An application developer launches the AA platform in order to design a multi-agent application. The main GUI of the development environment is provided by the Agent Factory (AF), a specifically designed agent, whose role is to collect all required information from the agent application developer regarding the definition of the types of agents involved in the MAS, the types of behaviours of these agents, as well as the ontology they share with each other. For this purpose, Agent Academy provides a Protégé–2000 front-end. 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 interoparate with the *Agent-Training Module* (ATM), which is responsible for inserting a specific decision model into them. The latter is produced by performing DM on data entered into Agent Academy as XML documents or as datasets stored in a database. This task is performed by the *Data Mining Module* (DMM), another agent of AA, whose task is to read available data and extract decision models, expressed in PMML format.

AA hosts a database system for storing all information about the configuration of the new created agents, their decision models, as well as data entered into the system for DM purposes. The whole AA platform was created as a MAS, which is executed upon JADE.

3.2 Developing Multi-agent Applications

The main GUI of the development platform (Agent Factory) consists of a set of graphical tools, which enable the developer to carry out all required tasks for the design and creation of a MAS, without any effort for writing even a single line of source code. In particular, the Agent Factory comprises the Ontology Design Tool, the Behaviour Type Design Tool, the Agent Type Definition Tool, and the MAS Creation Tool.

3.3 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 that are created with Protιgι 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.

3.4 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.

Figure 2 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.

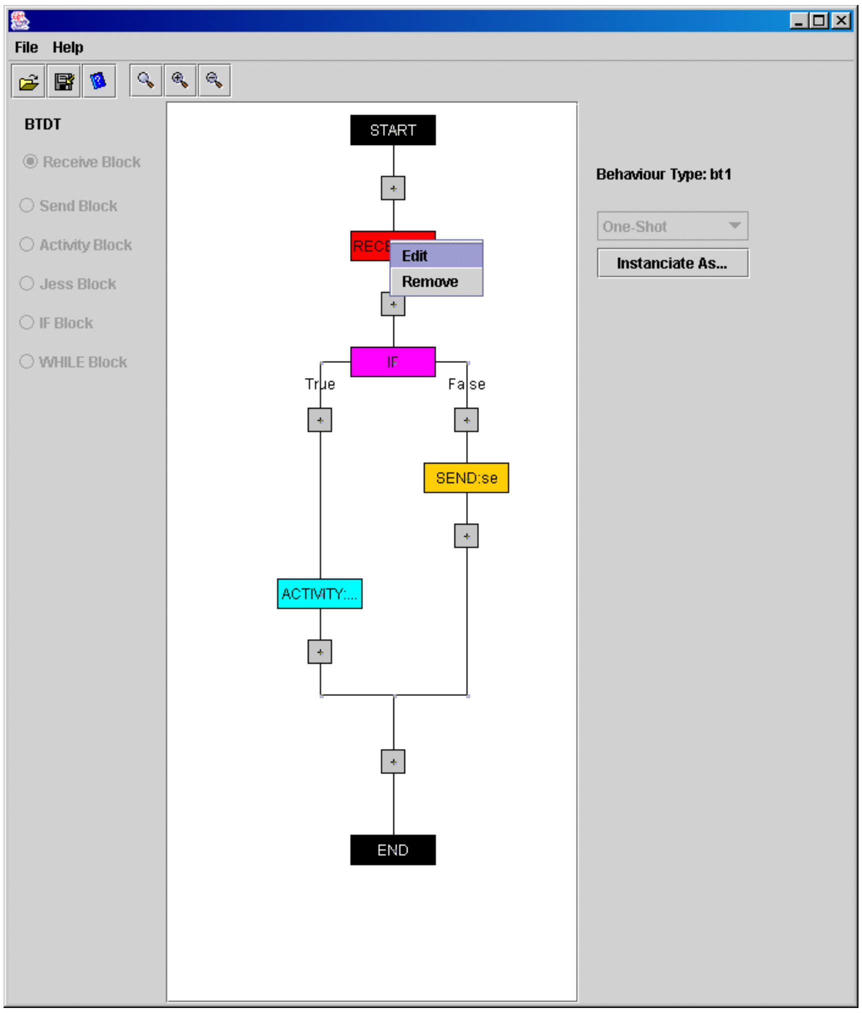


Fig. 2. Creating the Behaviour of an agent through the Behaviour Design Tool

3.5 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.

3.6 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 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 [17]. Thus, the lack of tools that concern agent reasoning issues in most high-level software design approaches is excused when we examine these approaches from a pure software-engineering point of view. Moreover, 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 Fig. 3. 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.

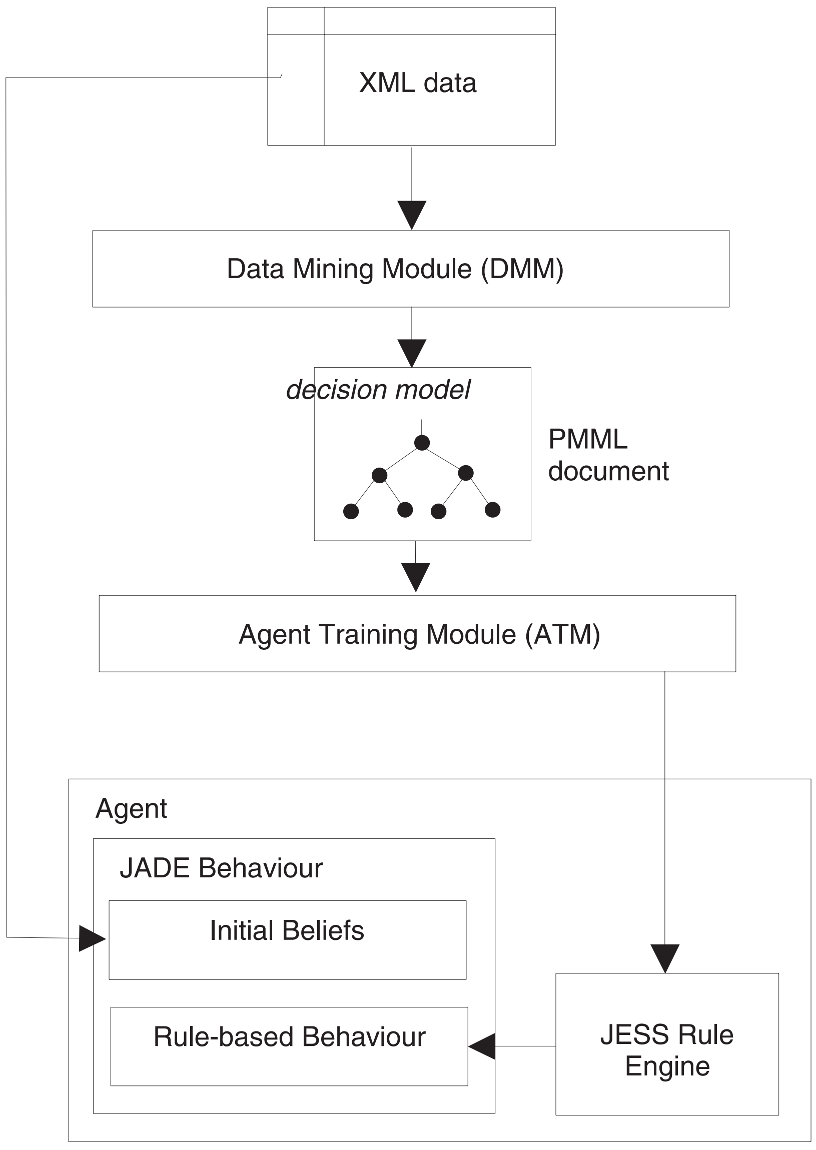


Fig. 3. Diagram of the agent training procedure

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) [11] 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 [18], a robust mechanism for executing rule-based reasoning. Finally, the execution of the rule engine becomes part of agent's behaviour.

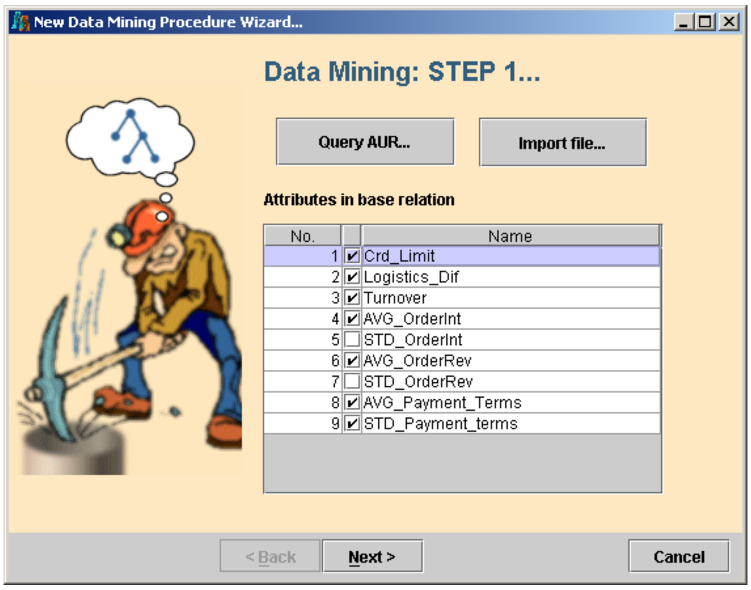
As shown in Fig. 3, an agent that can be trained through the provided infrastructure encapsulates two types of behaviours. The first is the basic initial behaviour predefined by the AF module. This may include a set of class instances that inherit the Behaviour class defined in JADE [5]. The initial 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 Mining Background Data for Creating Decision Models

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 [19]. From our experience with the application of our framework to an industrial scenario about supply chain management [20], 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.

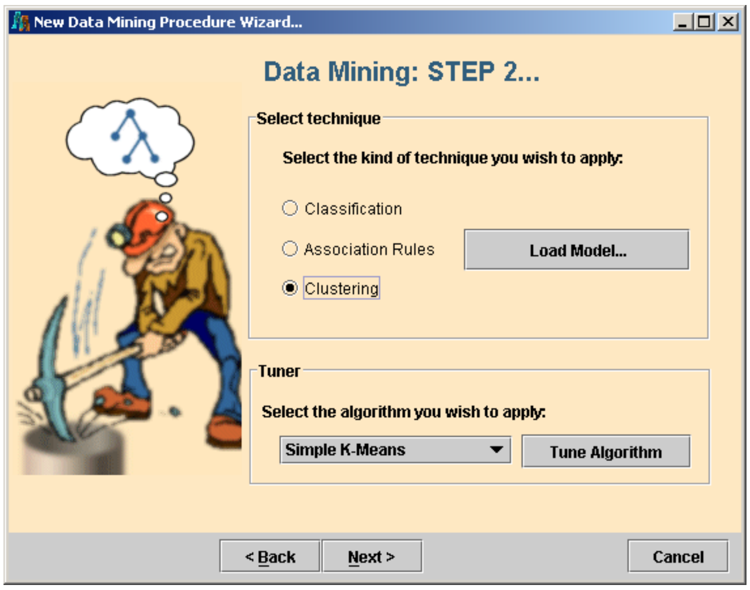
In the initial phase of the DM procedure, the developer launches the GUI-based wizard depicted in Fig.4(a) and specifies the data source to be loaded and the *agent decision attributes* that will be represented as internal nodes of the extracted decision model. In Fig.4(b) the developer selects the type of the DM technique from a set of available options. In order to clarify the meaning of agent decision attributes, let us consider the decision model in Fig.5. A certain decision is made when some or all input attributes are satisfied. In Fig. 5 we see an input



Data Mining: STEP 1...

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



Data Mining: STEP 2...

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

Fig. 4. The two first steps of the DMM wizard

vector of M attributes and an output vector with N attributes, which comprises the overall decision that an agent makes. One part of agent decision attributes is identical to the set of inputs that an agent receives, while the remaining part represents the outputs (decision nodes) of the agent.

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 [21] 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 [22].

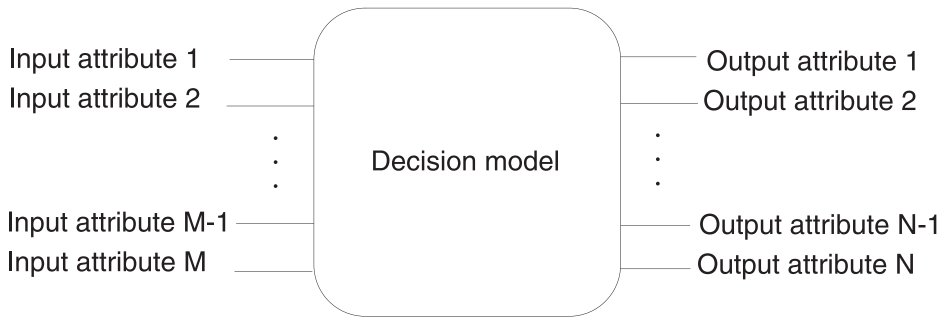


Fig. 5. Input and output attributes in a decision model

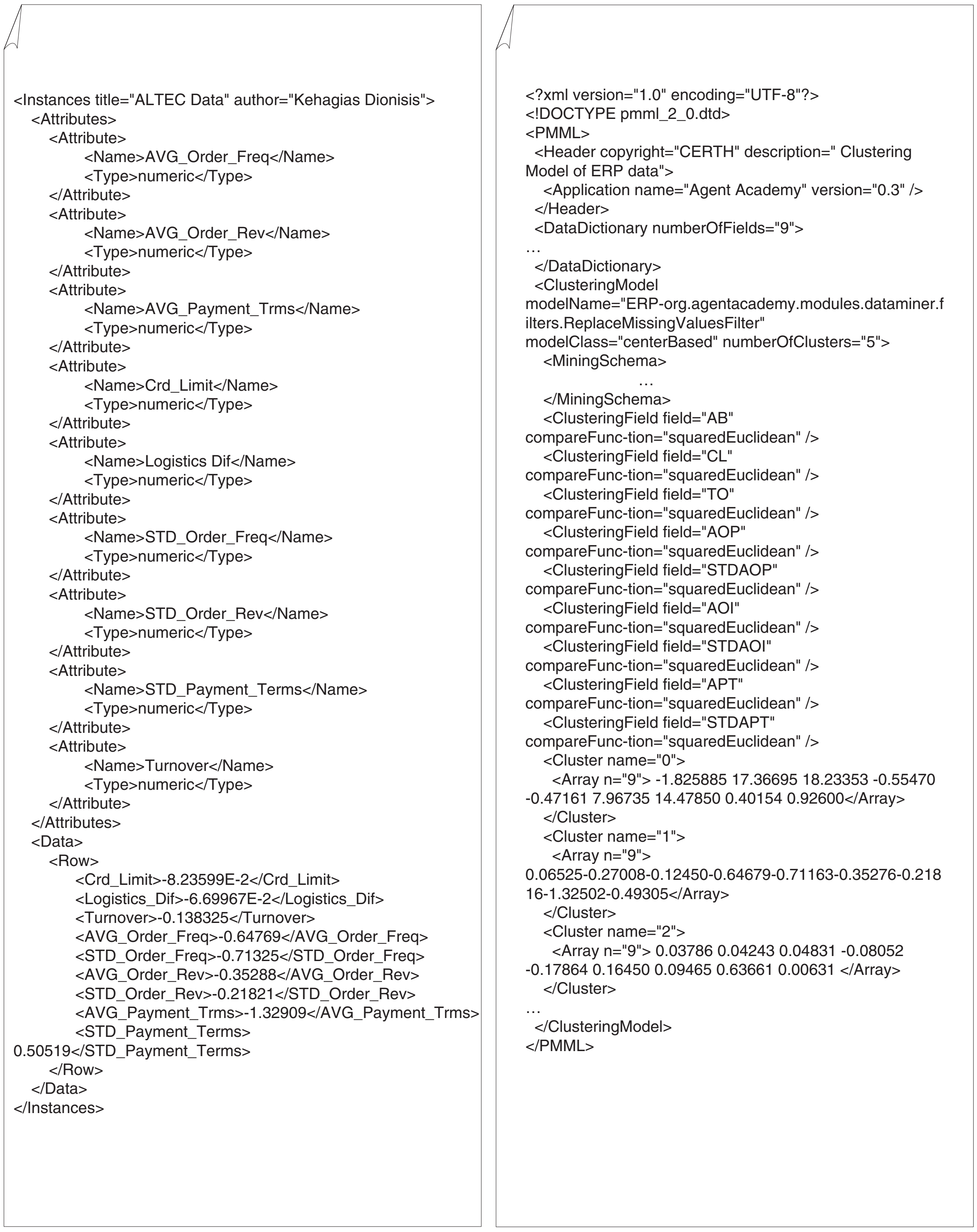
In Fig.6(a), we present an XML document, while the respective PMML output, which represents a cluster-based decision model, is shown in Fig.6(b). In order to generate the PMML output, we used the K-means algorithm to perform clustering on selected attributes described in the XML document. The dataset illustrated in Fig.6(a) comes from the design of the formerly mentioned MAS [20] about supply chain management. The XML document concerns customer-related data. The PMML output shown in Fig.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.

4.2 Embedding Intelligence into Agents

We saw in Fig.2 that 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 Java source code and instantiates 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



```
<Instances title="ALTEC Data" author="Kehagias Dionisis">
   <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>
          <STD_Order_Freq>-0.71325</STD_Order_Freq>
          <AVG_Order_Rev>-0.35288</AVG_Order_Rev>
          <STD_Order_Rev>-0.21821</STD_Order_Rev>
          <AVG_Payment_Trms>-1.32909</AVG_Payment_Trms>
          <STD_Payment_Terms>
0.50519</STD_Payment_Terms>
      </Row>
   </Data>
</Instances>
```

```
<?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.f
ilters.ReplaceMissingValuesFilter"
modelClass="centerBased" numberOfClusters="5">
    <MiningSchema>
                    …
    </MiningSchema>
    <ClusteringField field="AB"
compareFunc-tion="squaredEuclidean" />
    <ClusteringField field="CL"
compareFunc-tion="squaredEuclidean" />
    <ClusteringField field="TO"
compareFunc-tion="squaredEuclidean" />
    <ClusteringField field="AOP"
compareFunc-tion="squaredEuclidean" />
    <ClusteringField field="STDAOP"
compareFunc-tion="squaredEuclidean" />
    <ClusteringField field="AOI"
compareFunc-tion="squaredEuclidean" />
    <ClusteringField field="STDAOI"
compareFunc-tion="squaredEuclidean" />
    <ClusteringField field="APT"
compareFunc-tion="squaredEuclidean" />
    <ClusteringField field="STDAPT"
compareFunc-tion="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">
0.06525-0.27008-0.12450-0.64679-0.71163-0.35276-0.218
16-1.32502-0.49305</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>
…
  </ClusteringModel>
</PMML>
```

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

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. Currently, a beta version exists, which is not yet publicly available. The first stable implementation of AA is planned to come out on July 2003, as an open-source product. Our near future work involves the finalization of the integration process for 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, respectively.

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 Consortium: Agent Academy. (2000) Available at: http://agentacademy.iti.gr.
3. 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
4. Foundation for Intelligent Physical Agents: FIPA Developer’s Guide. (2001) Available at: http://www.fipa.org/specs/fipa00021/.
5. Bellifemine, F., Caire, G., Trucco, T., Rimassa, G.: JADE Programmer’s Guide. (2001) Available at: http://sharon.cselt.it/.
6. Wooldridge, M.J., Jennings, N.R., Kinny, D.: The gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems **3** (2000) 285–312
7. Foundation for Intelligent Physical Agents: FIPA Communicative Act Library Specification. (2001) Available at: http://www.fipa.org/specs/fipa00037/.
8. Foundation for Intelligent Physical Agents: FIPA SL Content Language Specification. (2002) Available at: http://www.fipa.org/specs/fipa00008/.

9. Foundation for Intelligent Physical Agents: FIPA ACL Message Structure Specification. (2002) Available at: http://www.fipa.org/specs/fipa000037/.
10. Noy, N.F., Sintek, M., S., D., Crubezy, M., Fergerson, R.W., Musen, M.A.: Creating semantic web contents with Protégé–2000. IEEE Intelligent Systems **16** (2001) 60–71
11. Data Mining Group: Predictive Model Markup Language Specifications (PMML), ver. 2.0. (2002) Available at: http://www.dmg.org.
12. Nwana, H., Ndumu, D., Lee, L., Collis, J.: ZEUS: A tool-kit for building distributed multi-agent systems. Applied Artifical Intelligence Journal **13** (1999) 129–186
13. Gutknecht, O., Ferber, J.: Madkit: A generic multi-agent platform. In: 4th International Conference on Autonomous Agents, Barcelona, Spain (2000)
14. Suguri, H., Kodama, E., Miyazaki, M., Nunokawa, H., Noguchi, S.: Madkit: A generic multi-agent platform. In: Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents, Montreal, Canada (2001)
15. Jeon, H., Petrie, C., Cutkosky, M.: JATLite: a java agent infrastructure with message routing. IEEE Internet Computing **4** (2000) 87–96
16. 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)
17. Tveit, A.: A survey of agent-oriented software engineering. In: Proceedings of the NTNU Computer Science Graduate Student Conference, Norwegian University of Science and Technology, Trondheim, Norway (2001)
18. Friedman-Hill, E.: Java Expert System Shell (JESS). Sandia National Laboratories. (2002) Available at: http://herzberg.ca.sandia.gov/jess.
19. 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)
20. Symeonidis, A.L., Kehagias, D., Koumpis, A., Vontas, A.: Open source supply chains. In: 10th International Conference on Concurrent Engineering (CE-2003), Workshop on intelligent agents and data mining: research and applications, Madeira, Portugal (2003)
21. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, San Francisco, CA (2000)
22. Athanasiadis, I.N., Kaburlasos, V.G., Mitkas, P.A., 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)