Embedding data–driven decision strategies on software agents: The case of a Multi–Agent System for Monitoring Air–Quality Indexes

I. N. Athanasiadis & P. A. Mitkas
Informatics and Telematics Institute, Thermi, Thessaloniki, Greece
and
Aristotle University of Thessaloniki, Thessaloniki, Greece

G. B. Laleci & Y. Kabak
Middle East Technical University, Ankara, Turkey

ABSTRACT: Agent Academy[1] (AA) is a software tool for deploying and training multi–agent communities, as it supports the design, creation and deployment of multi–agent systems (MAS). Even more, agents created with AA are equipped with a data-driven inference engine, and have the ability of training and retraining, through the AA's Agent Training Module. In the present paper, Agent Academy framework is presented. Agent Academy's advantages for creating agent-based intelligent software applications are revealed as the development of a Multi-Agent System for monitoring Air-Quality Indexes (O3RTAA) is described.

1 INTRODUCTION

1.1 *Multi-Agent Systems for Creating Intelligent Applications*

In the last decade, autonomous agents were introduced as a powerful metaphor for building software applications. Usually, agents are not developed as "stand-alone" applications; rather they are implemented to act within communities, called Multi-Agent Systems (MAS). Agent Technology is mounted on the principles of Concurrent Engineering (Agha 1986, Agha & Hewitt 1988, Agha et al.1993), as each one of the agents has its own thread of control.

MAS applications have been deployed in many application domains, such as: manufacturing, process control, telecommunication systems, air traffic control, traffic and transportation management, information filtering and gathering, electronic commerce, business process management, entertainment and medical care (Wooldridge & Jennings 1999).

Agent paradigm in building Intelligent Applications is summarized by Jennings et al. (1998) as follows: "A MAS can be defined as a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver. These problem solvers –agents– are autonomous and may be heterogeneous in nature. The characteristics of MAS are:

a. each agent has incomplete information, or capabilities for solving the problem, thus each agent has a limited viewpoint;
b. there is no global system control;
c. data is decentralized; and
d. computation is asynchronous."

In the aforementioned context, agent-based solutions have proven to be suitable for building intelligent applications following the concurrent engineering paradigm: While an agent has a view of its environments and based on its perceptions is able to decide on appropriate actions, when situated in a community of concurrently working agents propagatively contributes in the achievement of the MAS common goals, which are usually broader.

1.2 *Data Mining Use for Extracting Inference Models*

Chen (1999) states "The interplay between knowledge reasoning and data retrieval can be achieved by viewing retrieval as an extreme of reasoning and vice-versa". Based on the popular model of analogy, data-driven reasoning models, such as Decision Trees or Association Rules, could prove to be valuable in domains, where deductive logic is not applicable.

Furthermore, in deduction only the logical form of the argument needs to be considered; whereas in induction information about the world must be added in order to show that the conclusion follows with some degree of probability (Yezzi 1992). In this manner, the inductive logic seems to be more suit-

[1] Agent Academy: A Data Mining Framework for Creating Intelligent Agents, is partially funded by the European Commission, under the 5th Framework for Research and Development (IST–2000–31050).

able for agent-based solutions, as agents have a viewpoint on its world. The data mining approach introduces a set of tools and methodologies for discovering patterns. These patterns could be decision trees, association rules, neural networks, etc. The extracted patterns can be implemented as a knowledge model constituting an inference engine for taking decisions. This approach has been adopted by Agent Academy for training intelligent agents.

1.3 *Coupling MAS with DM results*

This paper describes the procedure followed by the Agent Academy (AA) project for embedding data-driven reasoning models on software agents in order to empower the latter with domain-specific intelligence. More specifically, the case of deploying a MAS for monitoring air-quality indexes is presented.

In Section 2 the Agent Academy framework is presented in brief and in Section 3 the explanatory case, named O3RTAA, is presented. The procedure for building the O3RTAA multi-agent system, with the use of Agent Academy follows in Section 4, while in Section 5 the agent community training is discussed. Finally, experiences from the Agent Academy project are underlined and some conclusions are made.

2 AGENT ACADEMY

2.1 *The AA architecture*

Agent Academy (AA) is a framework for training intelligent agents using data mining techniques (Agent Academy Consortium 2000, Mitkas et al. 2002) More specifically, Agent Academy is an integrated environment for embedding and improving intelligence in newly created agents through the use of Data Mining techniques performed on data derived from monitoring agent data and agent behavior. Agent Academy is a training facility that supports: a. the creation of agents with limited initial referencing capabilities, and b. the training of these agents in order to augment their intelligence efficiently, according to user specifications and preferences.

Agent Academy platform is comprised out of four modules:

a. the Agent Factory, for building (untrained) agents;
b. the Agent Use Repository, which stores agent-data;
c. the Data Miner (DM), that extracts knowledge from AUR’s data;
d. the Agent Training Module (ATM), which is responsible for training agents.

The Agent Academy architecture is shown in Figure 1. Agent Academy procedure for creating Multi-Agent Systems starts by the definition of the Agent Ontologies with the help of the ***Ontology Design Tool***. The information flow of each agent behavior is defined through the ***Behavior Type Design Tool***. In the platform, it is possible to design generic agent templates that can be further used while designing different multi-agent systems. Finally with the help of the ***Scenario Design Tool***, the interactions between the agents are defined, the specific agent instances are created and the Multi Agent System starts operating.

This work focuses on the AF functionalities for creating the agent community and the DM–ATM functionality for training intelligent agents. More specifically, the procedure for embedding intelligence extracted with Data Mining techniques on Agents will be discussed. The Agent Training procedure is described in Sections 4 & 5.

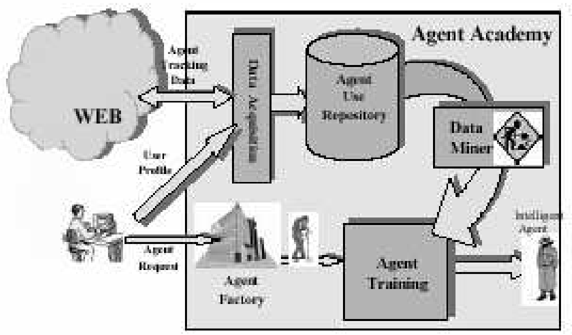

Figure 1. The Agent Academy architecture.

2.2 *Technologies incorporated*

Agent Academy adopts a bouquet of state-of-the-art technologies including:

a. JADE platform (for agent creation)
b. JESS engine (for rule execution)
c. Protégé (for ontology design and specification)
d. WEKA data mining tool (for knowledge extraction)
e. XML (for internal data exchange)
f. PMML (for knowledge model representation)
g. PostgreSQL RDBMS (for data and meta–data storage)
h. JMI (for meta-repository implementation).

Additionally, it should be mentioned that Agent Academy agents are compliant with FIPA specifications.

The reader can refer to these technologies in Grosso et al. (1999), Witten & Frank (1999), Bellifemine et al. (2000), FIPA (2000), Data Mining Group (2001), Friedman-Hill (2003).

3 O3RTAA: AN AGENT – BASED SYSTEM FOR MONITORING AIR-QUALITY INDEXES

3.1 *Introduction*

In this section the O3RTAA multi-agent system for monitoring air-quality indexes in real-time is presented. The O3RTAA system will be deployed by IDI-EIKON, Spain, and will be installed in the Mediterranean Centre for Environmental Studies Foundation (CEAM), in Valencia, Spain. The main goal of O3RTAA is to operate in a “live” environment and perform all appropriate tasks for detecting, analyzing and triggering ozone alarms to all concerned stakeholders, according to different profiles.

Several agents co-operate concurrently in a distributed agent society, in order to monitor both meteorological and air quality attributed and thus, evaluate air quality. The O3RTAA system is structured in three agent layers, shown in Figure 2. Each one of the layers undertakes the responsibility to achieve one of the system’s common goals.

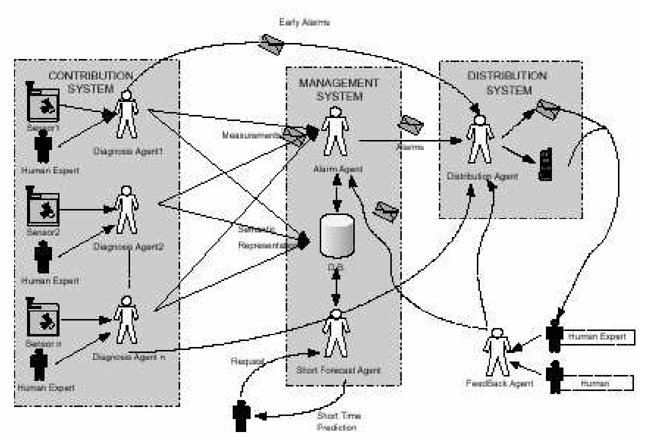

Figure 2. The O3RTAA System Architecture.

The first layer is the *Contribution Layer*. This part of the system is responsible for the acquisition and conditioning of data captured automatically by field sensors. Measurement validation, early alarm identification, sensor malfunction identification and qualitative estimation of the missing or erroneous values are the main goals of this layer.

The second layer is the *Management Layer*. In this layer the task is to analyze the data and fire the appropriate alarms. Additionally, the measurements are properly stored in the database.

The third layer is the *Distribution Layer*, which is responsible for sending the corresponding alarms to the users registered in the service, according to their profiles.

A set of agents in each layer is confronted with the task to achieve the respective goals. Several agent instances cooperate in each layer. This issue is discussed in depth in the following paragraph.

3.2 *Agent System Architecture*

Several agent types co-operate concurrently in O3RTAA system layers to achieve its goals. More specifically, the agent types deployed are the followings:

a. **Diagnosis Agents** (DA).
b. **Alarm Agents** (ALA).
c. **Short Prediction Agents** (SPA).
d. **Distribution Agents** (DIA).
e. **Feedback Agents** (FA).
f. **Database Agents** (DBA).

Each one of the Diagnosis Agents is devoted in monitoring a specific meteorological or air quality attribute, i.e. NO_2, NO_x, O_3, etc. Moreover, DA is responsible for ensuring the efficient operation of respective sensor. In case of a sensor breakdown, DA is responsible for predicting the missing value(s). Several DA instances are activated in the Contribution Layer, each one of which handles data coming from one sensor.

Alarm Agents evaluate the inputs and decide whether an alarm should be triggered or not. Short Prediction Agents take under account the current and past measurements and try to identify how air quality will evolve, based on certain patterns.

Distribution Agents are in charge of delivering alarms selectively, while Feedback Agents deliver users’ response on an alarm.

Finally, the Database Agent is in charge of delivering accurate, validated data to the measurements database.

In Section 4, the procedure for deploying the application with the use of Agent Academy platform is presented. Mainly, we concentrate on the deployment of the Contribution Layer.

4 DEPLOYING O3RTAA USING AGENT ACADEMY

Agent Academy Platform provides an easy way for deploying Multi-Agent Systems without any coding effort with the help of Agent Factory Module. Through a set of graphical tools, it is possible to define all the necessary details to allow the programmer design and create a Multi Agent System, either from scratch, or by making use of existing applications. The created agents have the ability to communicate the AA components such as Agent Training Module, Agent Factory, and reporting to the Agent Use Repository.

The Agent Factory provides a set of tools to enable these functionalities. More specifically, the Ontology Design Tool, the Behavior Type Design Tool, and the Scenario Design Tool are used for building the system. At the end, the multi-agent community is instantiated. In the following sections, the details of these functionalities will be presented while designing and deploying the O3RTAA System.

4.1 *Creating Agent Ontologies*

The first step in designing a Multi agent system, is defining the common language between the agents, i.e. the Ontologies. The Agent Factory provides an *Ontology Design Tool*, which help users adopt ontologies defined with the Protégé Tool (Grosso et al. 1999). RDFS files created with Protégé are saved in the Agent Use Repository for further use. As AA uses the JADE agent development environment, agent ontologies necessitate to be converted in to special JADE Ontology classes. Whenever an AA agent is created, the corresponding JADE Ontology classes are created after retrieving the respective Ontology files from the AUR, using a special tool, which compiles the RDFS ontology files into JADE Ontology classes.

For the O3RTAA system, we have defined an ontology specifying all of the necessary classes such as pollutants, measurements, meteorological stations and their attributes in terms of JADE Ontology concepts, predicates and agent actions. For example in Figure 3, the "StationInfo" is defined as a JADE Concept, and its attributes such as *calibration*, *stationName*, and *status* are defined.

```
<rdfs:Class rdf:about="&O3RTAA;StationInfo"
   rdfs:label="StationInfo">
  <rdfs:subClassOf rdf:resource="&O3RTAA;Concept"/>
</rdfs:Class>


<rdf:Property rdf:about="&O3RTAA;calibration"
a:maxCardinality="1" rdfs:label="calibration">
  <rdfs:domain rdf:resource="&O3RTAA;StationInfo"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>


<rdf:Property rdf:about="&O3RTAA;stationName"
   a:maxCardinality="1"
   rdfs:label="stationName">
  <rdfs:domain rdf:resource="&O3RTAA;StationInfo"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
 </rdf:Property>


<rdf:Property rdf:about="&O3RTAA;status"
   a:maxCardinality="1"
   rdfs:label="status">
  <rdfs:domain rdf:resource="&O3RTAA;StationInfo"/>
  <rdfs:range rdf:resource="&rdfs;Literal"/>
</rdf:Property>
```

Figure 3. A part of the O3RTAA Ontology

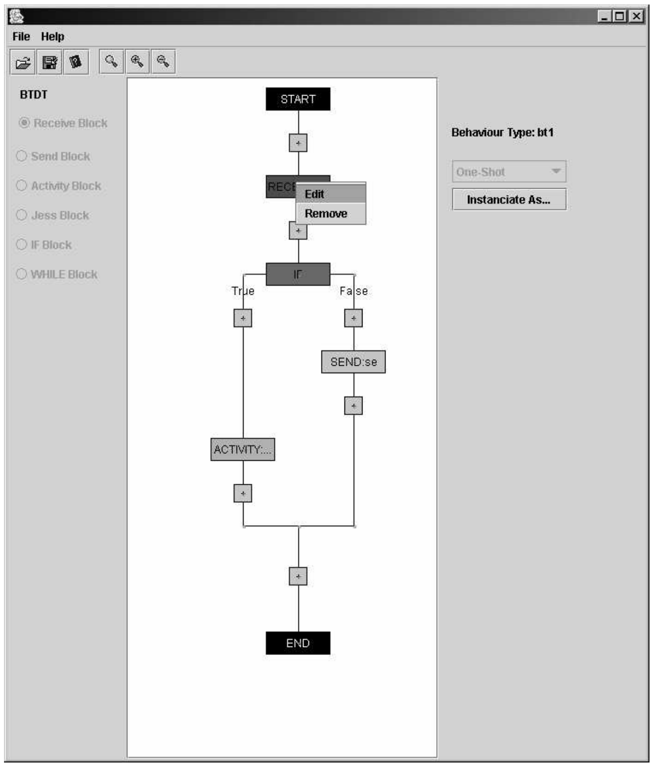

Figure 4. Behavior Design Tool

4.2 *Creating Behavior Types*

Using the Behavior Type Design Tool provided, it is possible to define generic behavior templates. Agent behaviors 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 behaviors can be modeled as *cyclic* or *one-shot* behaviors of the JADE platform. These behavior types are generic templates that can be configured to behave differently; only the structure of the flow is defined, the configurable parameters of the application inside the behavior, as well as the contents of the messages will be specified using the Scenario Design Tool while the behaviors are specialized according to the domain.

In order to explain the functionalities of this tool, we will go over the design process of the *Diagnosis Behavior Type*.

The first action in the Diagnosis Behavior is receiving the measurement from the respective Agent, so using the panel presented in Figure 4, a receive block is added to the flow of the agent behavior.

After receiving the measurement, the Diagnosis Agent validates this data by checking its conformity to data-driven patterns, extracted from historical data. The rules have been previously generated by the Data Miner Module, and the corresponding Decision Structure has been registered to the Agent Use Repository. In order to execute these rules and derive a decision, an inference engine has to be added to the flow of the behavior, specifying the decision structure that will be used by this inference engine. Hence, the initial rules are loaded to the agent, containing the Validation Decision Model. However this model (i.e. rules) may be updated in the future by the Data Miner Module, and loaded to the agent by the Agent Training Module; in this way the agent can adapt to the changing aspects of its environment, while running. (This is the case of retraining an agent using the AA platform).

An “if" block is added to the flow of behavior, to specify the actions that will be performed, depending on the validity of data. If the data is valid, the data is checked to see if it causes an “early alarm". Therefore an “action block" is added to the “true branch" of the “if" block. The parameters of the action block, i.e. the application that will be executed and the parameters of the application are specified. If an early alarm is produced by the activity, this alarm should be sent to the distribution agent (DIA). Therefore, an “if" block is added, the “if statement" is specified through the editor provided, and a “send block" is added to the ``true branch" of the “if" block. The performative and the ontology of the message are specified. However the receiver of this message is not set yet, since the exact agent instance that will receive the message will be assigned in the Scenario Design Tool.

If the inference engine decides that the measurement is not valid, then the agent tries to estimate the erroneous (missing) value of the measurement, by executing a second inference engine, containing the Estimation Decision Model. This Decision Model has as inputs measurement values that the agent handles in the past, and also the measurements of other Diagnosis Agents nearby, monitoring other related attributes. Hence, several send and receive blocks have been added to the flow of the behavior, for receiving those appropriate values. The Diagnosis Agent, after receiving the measurements, estimates the missing value, running the second inference engine, which uses a Decision Structure derived by the Data Miner Module. Hence, an inference engine is added to the flow and the necessary parameters are set using the editor.

Then an “activity block" is added after the “if" block to convert the measurements into their semantic representation. The application that will be executed in this activity block and its parameters are specified using the editor. Finally a “send block" is added in order to send this semantic representation to the Alarm Agent (ALA).

The other necessary behavior types are also designed in a similar fashion.

4.3 *Creating Agent Types*

The aim of the AF is to ease the development of multi agent systems, so after having defined certain behavior types, this tool is provided to create new agent types in order to be used later in the Scenario Design Tool. An agent type is in fact an agent plus a set of behaviors assigned to it. New agent types can be constructed from scratch or by modifying existing agent types. Agent types can be seen as templates that can be instantiated as agent instances while designing a scenario. For the O3RTAA system, we have defined six agent templates, one for each agent type. Namely: diagnosis, alarm, short prediction, distribution, feedback, and database agent templates.

While creating a Multi-Agent System, using the AA Scenario Design Tool, several instances of these agent types will be instantiated, having different values in their parameters. Each agent instance of the same agent type may have to deliver data from different sensor, or communicate with other agents, or run different decision models or access a different database and so on. These kinds of parameters are defined while deploying the MAS using the Scenario Design Tool.

4.4 *Deploying the Multi Agent System*

After designing the behavior types and the agent types, follows the deployment of the multi agent system. With the help of the Scenario Design Tool, all the agents running in that system are instantiated using the predefined agent templates. The receivers and senders of the messages in the behaviors of the agents are set, defining the interactions between the agents. Agent behaviors are also configured by setting all the necessary parameters, as inputs of the applications and content of the messages. For example, for the O3RTAA system, one *diagnosis agent* is initialized for each sensor. The DAs are configured for listening to different sensors located in different geographic locations.

After all the parameters are defined, the agent instances are initialized. Agent Factory creates *Default AA Agents*, which have the ability to communicate with AF, ATM and AUR. Then, the AF sends each of these agents the necessary ontologies, behaviors, and decision structures. Each agent parses the RDF Ontologies into JADE ontology classes using the tool provided, loads its behaviors, and starts operating.

5 TRAINING THE AGENT COMMUNITY

5.1 *Extracting data-driven Decision Models*

While describing the *Diagnosis Behavior Type* it has been mentioned that there are two kinds of data-driven decision blocks. The first one checks the validity of data and the second one estimates missing or erroneous measurements. These blocks are equipped with decision models extracted using the AA Data Miner Module . More specifically, the C4.5

algorithm (Quinlan 1993) for extracting decision trees was applied on historical data. The ONDA dataset supplied by CEAM, contained data from a meteorological station in the district of Valencia, Spain. More specifically, several meteorological attributes and air-pollutant values, along with validation tags, were recorded on a quarter-hourly basis during years 2000 and 2001. There are about 70,000 records in the volume.
The first set of experiments aimed to extract a decision model for evaluating an incoming measurement. The ONDA dataset was preprocessed in order to contain attributes as the current value of a specific pollutant and the corresponding validation tag, along with a set of previous values and measures. These measures are shown in Figure 3.

Quinlan's C4.5 algorithm for decision tree extraction was applied on the data. Data recorded in year 2000 were used as the training set and data recorded in 2001 were used as the test set. The pruning option for support 25% was selected after exhaustive experiments, producing a decision model with more than 99% accuracy for both training and test sets. The Confusion Matrix for the test set is shown in Figure 5.

The second decision structure extracted with the AA Data Miner Module is the one for estimating a missing measurement. Attributes, as missing measurement older values or values of other measurements at the same time were selected as inputs for this kind of decision models and are shown in Figure 4. From the ONDA dataset the invalid records or records with inconsistent history were excluded, remaining a volume of 12,000 records.

O3	The current ozone value
O3_30	The ozone value 30 min ago
O3_90	The ozone value 90 min ago
MinMax60	The difference between the maximum and the minimum ozone value in the last 60 min
MinMax150	The difference between the maximum and the minimum ozone value in the last 150 min
O3val	The corresponding validation tag (valid/erroneous)

Figure 3. Attributes used for the validation decision model

NO	The concurrent value of NO concentration
NO_2	The concurrent value of NO_2 concentration
NO_X	The concurrent value of NO_x concentration
TEM	The concurrent value of Temperature
HR	The concurrent value of Relative Humidity
O_3_15	The ozone value 15 min ago
O_3_30	The ozone value 30 min ago
O_3Class	The (missing) ozone value level (low/med)

Figure 4. Attributes used for the estimation decision model

Validation Decision Model

Records classified as :	valid	erroneous
No. records in class 'valid':	34,454	21
No. records in class 'erroneous':	63	420

Size of Decision Tree: 29 (15 Leaves)
Correctly classified records: 99.71%

Estimation Decision Model

Records classified as :	low	med
No. Records in class 'low':	9905	2,351
No. Records in class 'med':	752	4,384

Size of Decision Tree: 29 (15 Leaves)
Correctly classified records: 93.80%

Figure 5. Decision Model statistics.

```
<PMML>
 <DataDictionary numberOfFields="6">
 …
 </DataDictionary>
 <TreeModel modelName="…">
  <MiningSchema>
   <MiningField name="O3" usageType="active" />
   <MiningField name="O3val" usageType="predicted" />
    ………..
  </MiningSchema>
  <Node score="a">
   <SimplePredicate field="O3"
    operator="lessOrEqual" value="0" />
   <Node score="l"> <TRUE /> </Node>
    <Node score="a">
    <SimplePredicate field="O3"
      operator="greaterThan" value="0" />
    <Node score="a">
     <SimplePredicate field="O3"
      operator="lessOrEqual" value="164" />
     <Node score="a">
      <SimplePredicate field="MinMax60"
        operator="lessOrEqual" value="65" />
      <Node score="a"> <TRUE /> </Node>
     </Node>
     <Node score="a">
      <SimplePredicate field="MinMax60"
        operator="greaterThan" value="65" />
      <Node score="a">
       <SimplePredicate field="MinMax150"
         operator="lessOrEqual" value="185.9" />
       <Node score="a">
        <SimplePredicate field="O3_30"
          operator="lessOrEqual" value="127" />
        <Node score="a"> <TRUE /> </Node>
       </Node>
       <Node score="a">
        <SimplePredicate field="O3_30"
          operator="greaterThan" value="127" />
        <Node score="o"> <TRUE /> </Node>
       </Node>
      </Node>....
 </TreeModel>
</PMML>
```

Figure 6. An example PMML file

```
(defrule rule-0  ( O3 ?O3)
 (test ( <= ?O3 -99.9 ))
 => (store O3val l ))

(defrule rule-1  ( O3 ?O3)
 (test ( > ?O3 -99.9 ))
 ( O3 ?O3)
 (test ( <= ?O3 164 ))
 ( MinMax60 ?MinMax60)
 (test ( <= ?MinMax60 65 ))
 => (store O3val a ))
(defrule rule-2  ( MinMax150 ?MinMax150)
 (test ( <= ?MinMax150 185.9 ))
 ( O3_30 ?O3_30)
 (test ( <= ?O3_30 127 ))
 ( O3 ?O3)
 (test ( > ?O3 -99.9 ))
 ( O3 ?O3)
 (test ( <= ?O3 164 ))
 ( MinMax60 ?MinMax60)
 (test ( > ?MinMax60 65 ))
 => (store O3val a ))

(defrule rule-3  ( MinMax150 ?MinMax150)
 (test ( <= ?MinMax150 185.9 ))
 ( O3_30 ?O3_30)
 (test ( > ?O3_30 127 ))
 ( O3 ?O3)
 (test ( > ?O3 -99.9 ))
 ( O3 ?O3)
 (test ( <= ?O3 164 ))
 ( MinMax60 ?MinMax60)
 (test ( > ?MinMax60 65 ))
 => (store O3val o ))
```

Figure 7. The JESS rules generated

Once more, Quinlan's (1993) C4.5 algorithm for classification was used. Cross-folds training for 10 folds were performed. The decision tree extracted with pruning support 0,025 outperformed, providing accuracy greater than 90%. The Confusion Matrix is shown in Figure 5.

We have discussed elsewhere (Athanasiadis et.al 2003) a slightly different approach in estimating missing measurements using different types of decision models as Decision Trees, Neural Networks or Fuzzy Lattice Model.

The decision models extracted with the Data Miner are forwarded to the ATM using PMML 2.0 format and finally embedded on running agents as JESS Rules, as described in the following paragraph.

5.2 *Embedding Decision Models on Agents*

There are two circumstances in which a decision model is loaded to an AA-produced agent. In the first case, the case of **training**, a newly created agent is configured to have a decision structure and ATM loads the decision model into it by obtaining the decision structure content from Agent Use Repository. In the second case, the case of **retraining**, the AA agent already uses the decision structure and the Data Miner Module, comes out with an update of the later. In such case, DMM constitutes an ACL message containing the updated decision structure in PMML format (Data Mining Group 2000) and sends it to the Agent Training Module.

In both cases, after receiving the message either from AUR or DMM, ATM converts the PMML document into JESS rules and determines the AA agents that use the decision structure. Finally, the JESS rules are sent to the appropriate Agents via ACL messages and they insert (or update) their decision structures, accordingly. An example PMML decision model and its corresponding JESS rules are depicted in the following Figures 5, 6 respectively. More on Jess engine can be found in Friedman-Hill (2003).

In the training and retraining process, Agent Training Module plays an important role. ATM holds all the information (such as, the behavior ids of an agent, decision structures of an agent) about the agents to be trained. The information is used to administer the agents in the Agent Academy platform.

6 DISCUSSION

In the present paper the procedure followed for building a Multi-Agent System that monitors Air-Quality Indexes using the AA platform was presented. Procedures that in the traditional approach for deploying a MAS needed significant human efforts in the means of code programming, are now automated using graphical tools provided by the Agent Academy platform. Furthermore, the procedure of training and retraining agents from historical data, based on data-mining techniques is an advanced feature, incorporated in the AA platform.

An explanatory case, for building such a MAS that needs agents to be trained from historical data is the O3RTAA case, which was described in the present paper. Agents acting as mediators, deliver validated information to the appropriate stakeholders in distributed environment. The Diagnosis Agent for monitoring ozone measurements in the O3RTAA system designed, deployed and trained using the C4.5 algorithm.

Furthermore, the use of C4.5 algorithm came out with trustworthy decision models for validating incoming measurements and estimating the erroneous ones in the described application.

Future steps are concentrated in adding intelligence (i.e. inference engines) in the distribution module of the O3RTAA system, for delivering alarms in a more efficient manner.

ACKNOWLEDGEMENTS

Authors would like to express their acknowledgements to the Agent Academy Consortium for their valuable help. More specifically to the IDI-EIKON team for their efforts within Agent Academy project to deploy the O3RTAA system and CEAM for the provision of the ONDA dataset.

REFERENCES

Agha, G. (1986). ACTORS: A Model of Concurrent Computation in Distributed Systems. MA: The MITPress.

Agha, G. & Hewitt, C. (1988). Concurrent programming using actors. In Yonezawa, Y. & Tokoro, M. (Eds.), *Object-Oriented Concurrent Programming*, (pp. 37–57). MIT Press.

Agha, G., Wegner, P., & Yonezawa, A. (editors) (1993). Research Directions in *Concurrent Object-Oriented Programming*. Cambridge, MA: The MIT Press.

Agent Academy Consortium, the (2000). The Agent Academy Project. Available at: http://AgentAcademy.iti.gr

Athanasiadis, I. N., Kaburlasos, V. G., Mitkas, P. A., & Petridis, V. (2003). Applying machine-learning techniques on air quality data for real-time decision support. The First NAISO Conference on Information Technologies in Environmental Engineering, Gdansk, Poland.

Bellifemine, F., Poggi, A., & Rimassa, G. (2000). Developing multi-agent systems with JADE, In the *Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, Boston, MA. (Available at: http://jade.cselt.it).

Chen, Z. (1999). Computational Intelligence for Decision Support. CRC international series on computational intelligence. CRC Press.

Data Mining Group, the (2001). Predictive Model Markup Language Specifications (PMML), ver. 2.0 (Available at: http://www.dmg.org).

FIPA (2000), Foundation for Intelligent Physical Agents Specifications, (Available at: http://www.fipa.org/).

Friedman-Hill, E. J. (2003). Jess, The Expert System Shell for the Java Platform, version 6.1, CA, Sandia National Laboratories. (Available at: http://herzberg.ca.sandia.gov/jess/).

Grosso, W. E., et al.(1999). Knowledge Modeling at the Millennium, The Design and Evolution of Protege-2000. (Available at: http://protege.stanford.edu)

Jennings, N. R., Sycara, K., & Wooldridge, M. J. (1998). A roadmap of agent research and development. In *Autonomous Agents and Multi-Agent Systems 1*, (pp. 7–38)., Boston. Kluwer Academic Publishers.

Mitkas, P., et al. (2002). An agent framework for dynamic agent retraining: Agent academy. In Stanford-Smith, B., Chiozza, E., & Edin, M. (Editors), *Challenges and Achievements in e-business and e-work*, pages 757–764, Prague, Czech Republic.

Quinlan, J.R. (1993) C4.5 Programs for Machine Learning, Morgan Kaufmann series in machine learning, USA, Morgan Kaufmann publishers.

Witten, I. H. & Frank, E. (1999) Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, USA, Morgan Kaufmann publishers.

Wooldridge, M. J., & Jennings, N. R. (1999). Software engineering with agents: Pitfalls and pratfalls. *IEEE Internet Computin*g, 3(3), pages 20–27.

Woolbridge, M. (1997). Agent-based software engineering. In *IEE Proc. Software Engineerin*g, number 144(1), pages 26–37.

Yezzi, R. (1992). Defining deduction and induction. *In Practical Logic, Enrichment Study 12,* Mancato. G. Bruno & Co.