Application of Data Mining and Intelligent Agent Technologies to Concurrent Engineering

P. A. Mitkas, A.L. Symeonidis, D. Kehagias & I. Athanasiadis
Informatics and Telematics Institute, Thermi, Thessaloniki, Greece
and
Aristotle University of Thessaloniki, Thessaloniki, Greece

ABSTRACT: Software agent technology has matured enough to produce intelligent agents, which can be used to control a large number of concurrent engineering tasks. Multi-agent systems are communities of agents that exchange information and data in the form of messages. The agents' intelligence can range from rudimentary sensor monitoring and data reporting to more advanced forms of decision making and autonomous behavior. The behavior and intelligence of each agent in the community can be obtained by performing data mining on available application data and the respected knowledge domain. We have developed Agent Academy, a software platform for the design, creation, and deployment of multi-agent systems, which combines the power of knowledge discovery algorithms with the versatility of agents. Using this platform, we show how agents, equipped with a data-driven inference engine, can be dynamically and continuously trained. We also discuss a few prototype multi-agent systems developed with Agent Academy.

1 INTRODUCTION

The idea of using a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support has proven appealing. Concurrent Engineering (CE) is intended to lead the developer, from the outset, to consider all elements of the product lifecycle from concept through disposal, including quality control, cost, scheduling and user requirements. The Integrated Product Development (IPD) process that CE exploits is a philosophy that systematically employs a teaming of functional disciplines to integrate and concurrently apply all necessary processes to produce an effective and efficient product that satisfies customer needs. There is no checklist for implementing IPD because there is no one solution...each application can be unique.

Since IPD is, in fact, the splitting of a major deliverable into many, simultaneous tasks (Figure 1), CE can be thought of merely as multi-tasking, which has efficiently been dealt with the flourishing and establishment of object-oriented programming. Allowing changes with small computational cost and increasing versatility, autonomy and heterogeneity has made autonomous agents (state-of-the-art for objected-oriented programming) to be introduced as a powerful metaphor for building software applications. Usually, these agents are not developed as "stand-alone" applications; rather they are implemented to act within communities, called Multi-Agent Systems (MAS).

It is therefore safe to deduce, that since each of the agents in a MAS has its own thread of control, with its own beliefs and experiences, Agent technology can be considered to emerge from the principles of Concurrent Engineering (Agha 1986, Agha & Hewitt 1988, Agha et al.1993).

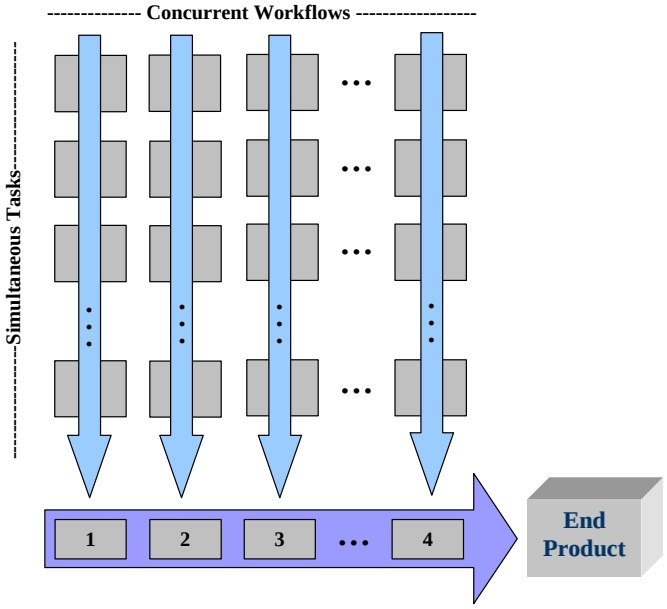

Figure 1. The process of Concurrent Engineering

In our attempt to justify our claim that a MAS can deal with the problem of CE efficiently, we had to use an agent development platform that could add some form of intelligence in the CE oriented application. Nevertheless, none of the existing development platforms has fulfilled our demand to provide enhanced capabilities in terms of the level of abstraction in the design and development process of agent-oriented applications (Nwana et.al. 1999,

Gutknecht et.al. 2000, Suguri et.al. 2001, Jeon et.al. 2000). A quite desirable effort was the creation of a software product that combines all widely used mainstream technologies in one tool. This is why we have developed Agent Academy (AA) (Agent Academy Consortium 2000, Mitkas et al. 2002), 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 (Bellifemine et al. 2000) infrastructure, ensuring a relatively high degree of FIPA compatibility, as defined in (FIPA 2000, Bellifemine et al. 2000). AA is itself a multi-agent system, whose architecture is based on the GAIA methodology (Wooldridge et. al 2000). 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 behavior types and attributes in a few minutes. Using AA, an agent developer can easily go into the details of the designated behaviors 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 (FIPA 2000, FIPA 2001, FIPA 2002), as well as the semantics, which can be defined by constructing ontologies with Protégé-2000 (Grosso et.al. 1999).

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) (Data Mining Group 2001) 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 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 3, a detailed presentation of the agent "training" mechanism is given. Section 4 introduces the three test case pilots of the platform, while section 5 concludes the paper and outlines future work.

2 THE AGENT ACADEMY 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:

1 the creation of new agents with limited initial reasoning capabilities;
2 the addition of these agents into a new MAS;
3 the determination of existing, or the creation of new behavior types for each agent;
4 the importation of ontology-files from Protégé-2000;
5 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:

– 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 2 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.

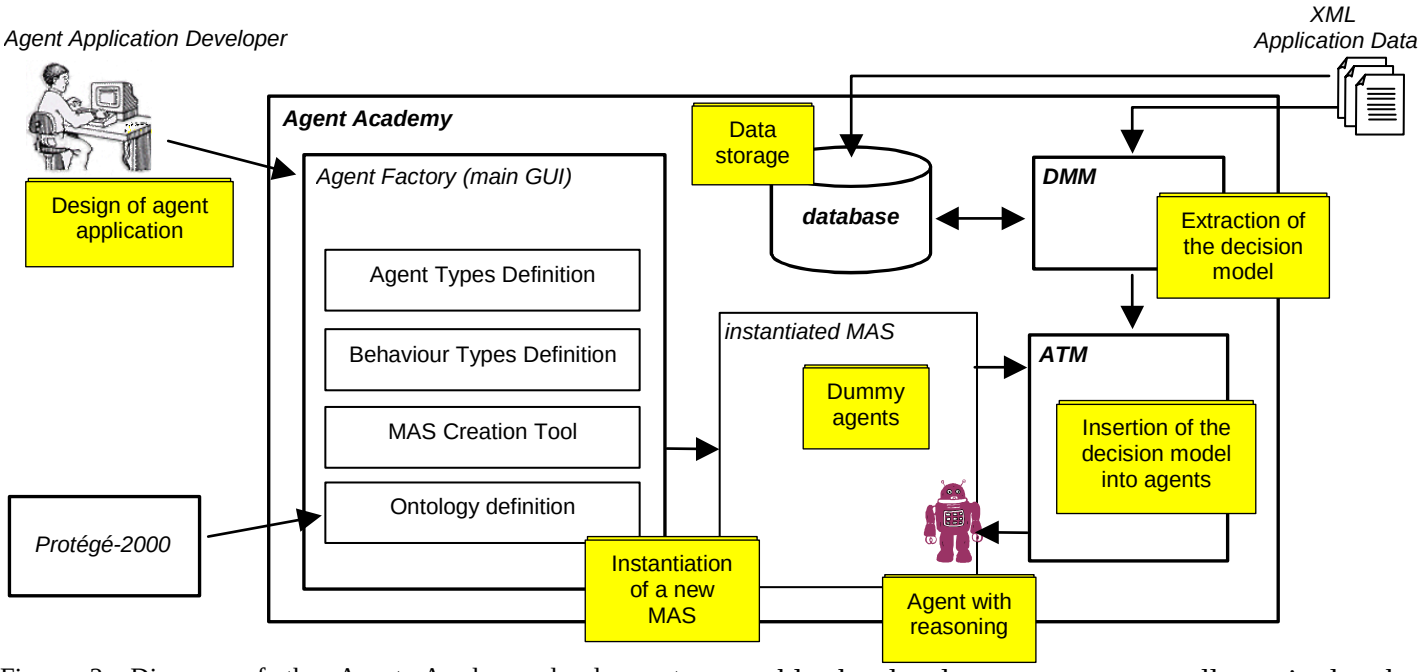

Figure 2. Diagram of the Agent Academy development framework

2.1 *Architecture*

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 behaviors 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 interoperate 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.

2.2 *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 Behavior Type Design Tool, the Agent Type Definition Tool, and the MAS Creation Tool.

2.2.1 *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 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.

2.2.2 *Creating Behavior Types*

The Behavior Type Design Tool assists the developer in defining 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 in different ways; the structure of the flow is the only process defined, while the configurable parameters of the application inside the behavior, as well as the contents of the messages can be specified using the MAS Creation Tool. It should be denoted that the

behaviors 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 behavior, 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.

2.2.3 *Creating Agent Types*

After having defined certain behavior 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 behaviors 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.

2.2.4 *Deploying a Multi Agent System*

The design of the behavior 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 behaviors 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, behaviors, and decision structures.

3 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 (Witten et. al. 2000). 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 figure 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.

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) 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 (Friedman-Hill 2003), a robust mechanism for executing rule-based reasoning. Finally, the execution of the rule engine becomes part of agent's behavior.

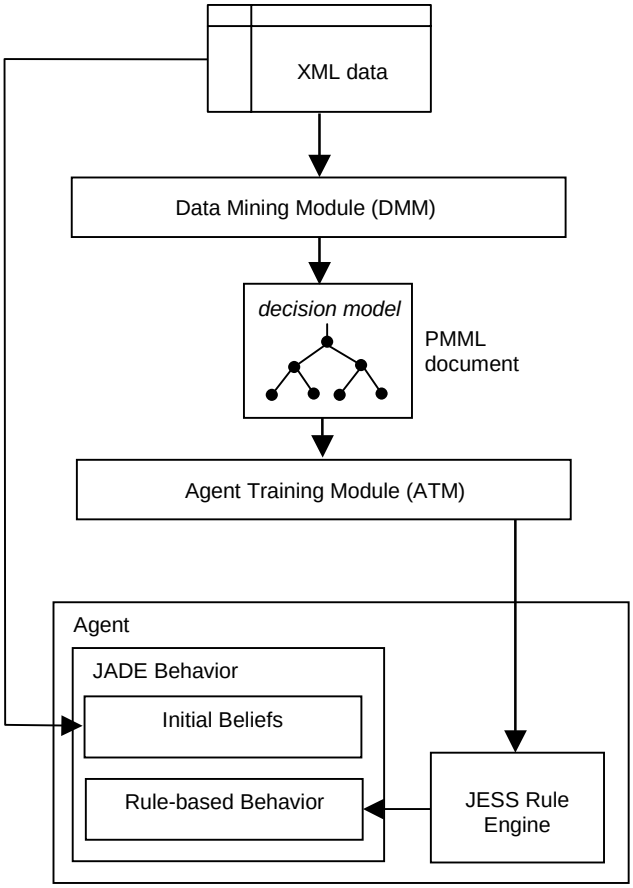

Figure 3. Diagram of the agent training procedure

As shown in figure 3, an agent that can be trained through the provided infrastructure encapsulates two types of behaviors. The first is the basic initial behavior predefined by the AF module. This may include a set of class instances that inherit the Behavior class defined in JADE. The initial behavior is created at the agent generation phase, using the Behavior Design Tool, as described in the previous section. This type of behavior characterizes all agents designed by Agent Academy, even if the developer intends to equip them with rule-based reasoning capabilities. This essential type of behavior includes the set of initial agent beliefs.

The second supported type of behavior is the rule-based behavior, which is optionally created, upon activation of the agent-training feature. This type of behavior 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.

3.1 *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 (Symeonidis et. al. 2002). From our experience with the application of our framework to an industrial scenario about supply chain management (Symeonidis et. al. 2003), 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 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 figure 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 figure 5. A certain decision is made when some or all input attributes are satisfied. In figure 5 we see an input 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 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 (Athanasiadis et. al. 2002).

3.2 *Embedding intelligence into agents*

The completion of the training process requires the translation of the DM resulted decision model into an agent-understandable format (Figure 2). 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.

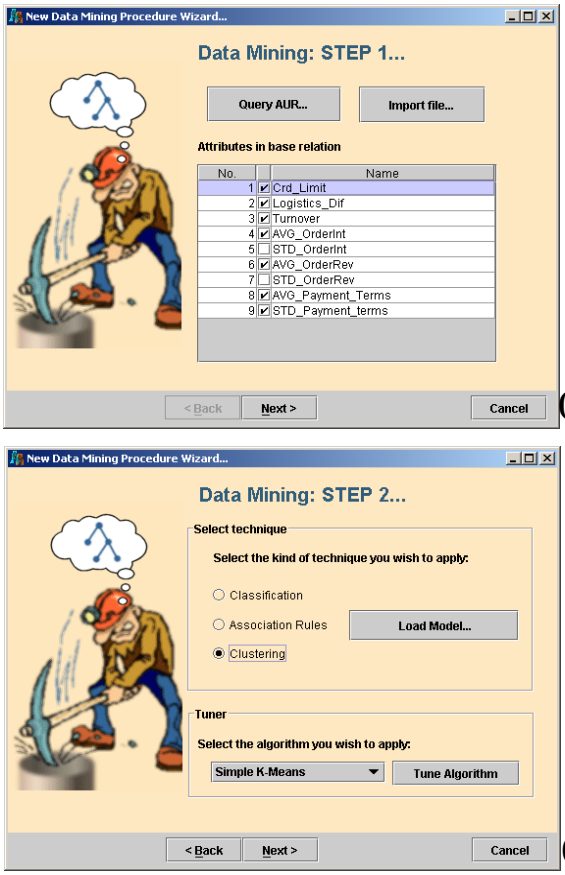

Figure 4. The two first steps of the DMM wizard

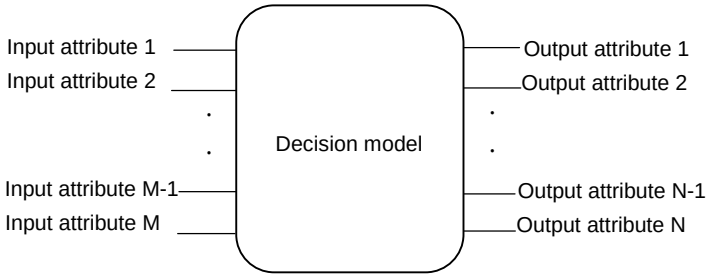

Figure 5. Input and output attributes in a decision model

4 DEVELOPED TEST CASES

The AA consortium is developing three test cases, in order to prove out the AA hypothesis. Through the test cases the platform usability and added value to the users- developer companies is revealed. The three test cases are deployed in the following domains:

- In-house Supply Chain Management
- Real-time environmental monitoring and assessment
- Web-based Distributed Service Management

The first test case scenario addresses issues concerning Supply Chain Management (SCM). This field is of great interest to a series of companies throughout Europe, as SCM depends on several output-significant variables. Based on an existing agent-based application for production planning, an agent-based SCM add-on on an ERP system is being developed.

The second test case scenario's purpose is to evaluate the use of AA trained agents in a real-time context. More specifically, the O3 RTAA system, a real-time Ozone Monitoring and Alarming application is under development.

Finally, the third test case scenario addresses web-based applications for distributed service management (DSM). A fast growing number of companies worldwide face the need for establishing a DSM system for matching their customers' needs in the after sales phase. An intelligent agent based application is proposed for rendering this kind of services.

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. 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.

6 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.

7 REFERENCES

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

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

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

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) (accepted for publication) available at: `http://danae.ee.auth.gr/en/pdf/itee2003.pdf`)

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)

Gutknecht, O., Ferber, J. (2000). Madkit: A generic multi-agent platform. In: 4th International Conference on Autonomous Agents, Barcelona, Spain

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.

Jeon, H., Petrie, C., Cutkosky, M. (2000). JATLite: a Java agent infrastructure with message routing. In: IEEE Internet Computing 4 (2) 87-96

Laleci, G. B., Kabak, Y., Dogac, A., Cingil I., Kirbas S., Yildiz A., Sinir S., Ozdikis O., Ozturk O. (2003). A Platform for Agent Behavior Design and Multi Agent Orchestration, Submitted to *IEEE Transactions on Knowledge and Data Engineeing.*

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.

Nwana, H., Ndumu, D., Lee, L., Collis, J. (1999). ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. In: Applied Artifical Intelligence Journal, Vol 13 (1) 129-186

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

Suguri, H., Kodama, E., Miyazaki, M., Nunokawa, H., Noguchi, S. (2001). Implementation of FIPA Ontology Service. In: Proceedings of the Workshop on Ontologies in Agent Systems, 5th International Conference on Autonomous Agents Montreal, Canada

Symeonidis A.L, Kehagias D., Koumpis A., Vontas A. (2003). 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 (accepted for publication)

Symeonidis, A.L., Mitkas, P.A., Kehagias, D. (2002). 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)

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

Witten, I.H., Frank, E. (2000). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann publishers, San Francisco, CA

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

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

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