In International Journal of Product Lifecycle Management, **2** (**2**):173-186, 2007.

Application of Data Mining and intelligent agent technologies to Concurrent Engineering

Pericles A. Mitkas* and
Andreas L. Symeonidis

Electrical and Computer Engineering Department,
Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
Fax: +30 2310 99 6398
and
Intelligent Systems and Software Engineering Laboratory,
Informatics and Telematics Institute/CERTH,
1st km Thermi-Panorama road, 57001,
Thermi, Thessaloniki, Greece
E-mail: mitkas@iti.gr E-mail: asymeon@iti.gr
*Corresponding author

Dionisis D. Kehagias

Intelligent Systems and Software Engineering Laboratory,
Informatics and Telematics Institute/CERTH,
1st km Thermi-Panorama road, 57001,
Thermi, Thessaloniki, Greece
E-mail: diok @iti.gr

Ioannis N. Athanasiadis

Istituto Dalle Molle di Studi sull' Intelligenza Artificiale,
CH-6928 Manna,
Lugano, Switzerland
E-mail: ioannis@idsia.ch

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 (MAS) 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 behaviour. The behaviour 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 (AA), a software platform for the design, creation, and deployment of MAS, which combines the power of knowledge discovery algorithms with the versatility of agents. Using this platform, we illustrate how agents, equipped with a data-driven inference engine, can be dynamically and continuously trained. We also discuss three prototype MAS developed with AA.

Keywords: Agent Academy; Concurrent Engineering; Data Mining; Intelligent agents.

Reference to this paper should be made as follows: Mitkas, P.A., Symeonidis, A.L., Kehagias, D.D. and Athanasiadis, I.N. (2007) 'Application of Data Mining and intelligent agent technologies to Concurrent Engineering', *Int. J. Product Lifecycle Management*, Vol. 2, No. 2, pp.173–186.

Biographical notes: Pericles A. Mitkas is a Professor of Electrical and Computer Engineering at the Aristotle University of Thessaloniki, Greece. He is also the Associate Director of the Informatics and Telematics Institute. His research interests include databases and knowledge bases, data mining, concurrent engineering, software agents, and bioinformatics. He received his Diploma of Electrical Engineering from Aristotle University of Thessaloniki in 1985 and an MSc and a PhD in Computer Engineering from Syracuse University, USA, in 1987 and 1990, respectively.

Andreas L. Symeonidis is a Lecturer with the Department of Electrical and Computer Engineering at the Aristotle University of Thessaloniki and a Research Associate with the Informatics and Telematics Institute in Thessaloniki, Greece. He received his diploma and PhD from the Department of Electrical and Computer Engineering at the Aristotle University of Thessaloniki. His research interests include software agents, data mining and knowledge extraction, intelligent systems, and evolutionary computing.

Dionisis D. Kehagias holds a Diploma and a PhD in Electrical and Computer Engineering from the Aristotle University of Thessaloniki, Greece and he currently works as an Associate Researcher at the Informatics and Telematics Institute in Thessaloniki, Greece. His research interests include intelligent agents and data mining.

Dr. Ioannis N. Athanasiadis holds a Diploma and a PhD in Electrical and Computer Engineering, both from the Aristotle University of Thessaloniki, Greece. Currently, he is a Researcher with the Dalle Molle Institute for Artificial Intelligence, in Lugano, Switzerland. His research interests include software engineering for ecoinformatics, ontologies and the semantic web, software agents, and knowledge engineering. In 2004, he was awarded the Student Prize of the International Environmental Modelling and Software Society.

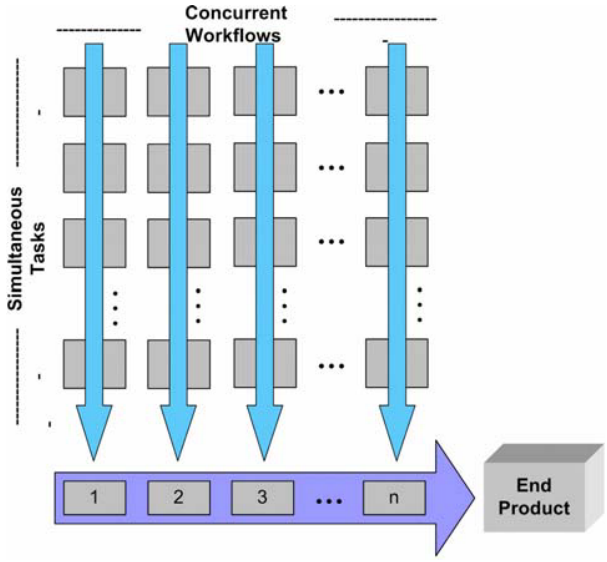
1 Introduction

The idea of using a systematic approach for the integrated, concurrent design of products and their related processes, including manufacturing and support has proven appealing. Concurrent Engineering (CE) is intended to guide the developer, through all the elements of a product's lifecycle, from concept to disposal, including quality control, cost estimation, scheduling, and user requirements. The Integrated Product Development (IPD) process, that CE embraces, is a philosophy that systematically employs a teaming of functional disciplines to integrate and concurrently apply all necessary processes, in order to efficiently engineer a product according to the customer needs. There are no guidelines for implementing IPD, because there is no single solution – each application can be unique. Since IPD is, in fact, the splitting of a major deliverable into multiple,

simultaneous tasks, real world CE design projects require the cooperation of multi-disciplinary design teams using sophisticated and powerful engineering tools (Shen, Norrie and Barthes, 2001).

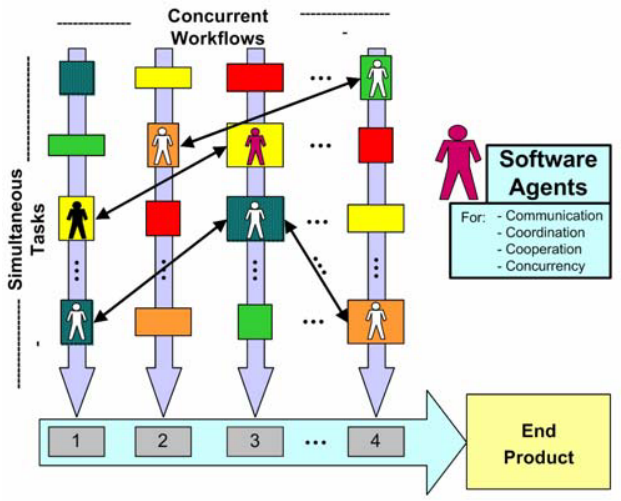
The process of CE is shown in Figure 1. The IPD process is comprised of a set of concurrent workflows executed horizontally leading to the implementation of a system. Each workflow consists of a set of simultaneous tasks executed vertically. The end of all tasks in a workflow signals the completion of one step in the horizontal execution process. From a computing perspective, CE has efficiently been serviced by the flourishing and establishment of object-oriented programming. The desirable characteristics of autonomy and heterogeneity, along with the need to implement changes at small computational cost and increased versatility, have led to the introduction of autonomous agents (state-of-the-art for objected-oriented programming) as a powerful metaphor for building software applications. Usually, these agents are not developed as 'stand-alone' applications; rather they are built to act within communities, called Multi-Agent Systems (MAS).

Figure 1 The process of concurrent engineering



Since each one of the agents in MAS has its own thread of control, with its own beliefs and cooperation primitives, agent technology can be considered to emerge from the principles of CE (Agha, 1986; Agha and Hewitt, 1988; Agha, Wegner and Yonezawa, 1988). The deployment of this programming paradigm leads to the mapping of the discrete, yet related, engineering tasks and subtasks of the CE design process into (generic) agents and agent behaviours. In Figure 2, agents undertake the responsibility of executing the vertical tasks. Each agent corresponds to a parallel process and the signalling required for the synchronisation of the tasks is implemented by messages exchanged among agents. The agent oriented perspective of the CE process shown in Figure 2 enables execution of tasks by exploiting the abilities of agents of cooperation and coordinated activity.

Figure 2 Deploying agent technology for Concurrent Engineering

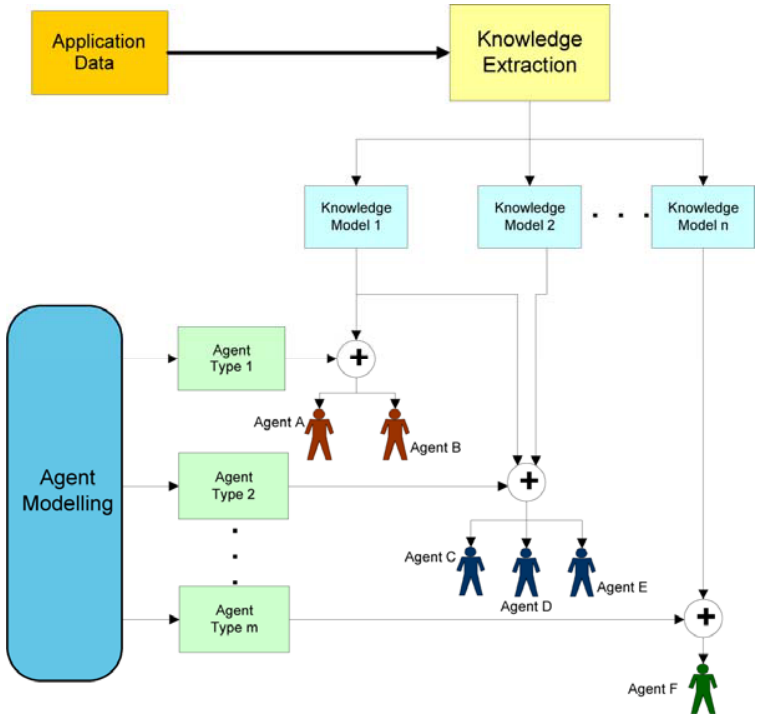


Let us consider the case of an organisation that has decided to adopt the agent-based approach and wants to transform their CE process into a MAS. First, through an agent modelling phase, the number and types of agents must be determined. If the existing CE process is well-defined and divided into separate subtasks, the assignment of tasks to the agents can be a straightforward exercise. The second step, however, that of determining the behaviour and reasoning engine of each agent, can be considerably more complex and, thus, costly. This step can be facilitated by the availability of historical data on prior decisions and actions of the organisation. As shown in Figure 3, knowledge extraction techniques can be applied on historical data in order to derive one or more knowledge models. Such models will provide the reasoning engines of agents in the form of decision trees or rule engines and they can be inserted 'as–is' or combined into agent types.

The confluence of agent modelling and knowledge extraction, as described in the figure, can be automated to become more efficient. This software development process can be especially exploited in environments with large amounts of periodically produced data, as is the typical environment of an Information Technology (IT)-supported enterprise today.

A number of agent-based approaches have been implemented (Cutkosky et al., 1993; McGuire et al., 1993; Petrie et al., 1994) indicating the need for an integrated agent development platform, which could consolidate the issues of agent design, communication, coordination, and control. Nevertheless, none of the existing development platforms has fulfiled this demand, i.e. 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 and Ferber, 2000; Jeon, Petrie and Cutkosky, 2000; Suguri et al., 2001). A quite desirable approach would be the creation of a software product that supports knowledge extraction and agent development, and combines several widely used mainstream technologies into a common environment. This is why we have developed Agent Academy (AA- Agent Academy on sourceforge, 2003; Mitkas et al., 2004), an integrated framework for constructing multi-agent applications and for embedding rule-based reasoning into agents, both at the design phase and at the runtime.

Figure 3 Combining agent modelling and knowledge extraction for the development of Multi-Agent Systems



This framework has been implemented upon the Java Agent Development (JADE) infrastructure (Jennings, Sycara and Wooldridge, 1998). AA is itself an MAS, whose architecture is based on the GAIA methodology (Wooldridge, 1997). It provides an integrated Graphical User Interface (GUI)-based environment that enables the design of both single- and multi-agent communities, using common drag-and-drop operations. This capability of AA enables agent application developers to build a whole community of agents with predefined behaviour types and attributes in short time. Using AA, an agent developer can easily go into the details of the designated behaviours of agents and precisely regulate their communication properties. These include the type and number of the Agent Communication Language (ACL) messages exchanged between agents, the performatives and structure of messages – conforming to FIPA specifications (FIPA, Foundation for Intelligent Physical Agents Specifications, 2000) – as well as the semantics, which can be defined by constructing ontologies with Protégé-2000 (Grosso et al., 1999).

The latter is a publicly available ontology-authoring tool. It is free, distributed under an open source license. Therefore, it allows direct access to its Application Programming Interface (API), which is written in Java. This was the main reason for using Protégé. Since AA, also distributed as an open source product, is written in Java, it can launch Protégé from inside its main application. Another reason for choosing Protégé is that today it is a widely supported tool forming the *de facto* standard for building ontologies.

The latest version of Protégé supports the latest specification of the Web Ontology Language (OWL).

It should also be noted that the AA framework approaches agent development not only from a pure software engineering perspective, but also deals with agent reasoning. For this purpose, it implements the training mechanism illustrated in Figure 3. Specifically, AA involves the use of Data Mining (DM) techniques for knowledge extraction. DM is applied on application data generated by the history of past design and production processes. The methodology developed within AA, results in the extraction of agent reasoning models associated with knowledge models (e.g. a decision tree, a set of clusters, etc.). On the other hand, appropriate agent types and behaviours are designed by the corresponding tools of AA. Bringing together knowledge models and agent types, the extracted models are embedded into specific agent instantiations. The applied DM techniques are, by definition, updateable as new data come into the AA repository. Thus, it is easy to update the reasoning models of agents, by performing agent 'retraining'.

The rest of 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 MAS. 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 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 behaviour types for each agent

4 the importation of ontology-files from Protégé-2000

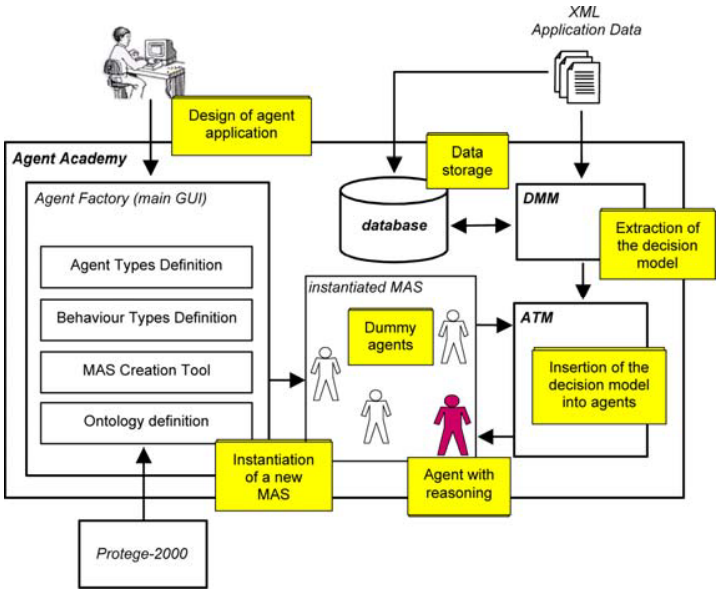
5 the determination of message recipients for each agent.

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 4 illustrates the AA main functional diagram, which represents the core components and the interactions between them. In the remaining sections, we discuss the AA architecture and explain the agent development process.

Figure 4 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 the required information from the agent application developer regarding the definition of the types of agents involved in MAS, the types of behaviours of these agents, as well as the ontology they share with each other. The initially created agents possess no referencing capabilities (denoted as '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 knowledge model into them. The latter is produced by performing DM on data entered into AA as XML documents or as datasets stored in a database. This task is performed by the Data Miner Module (DMM), another agent of AA, whose task is to read available data and extract knowledge models.

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 MAS, which is executed upon JADE.

2.2 Developing multi-agent applications

AF consists of a set of graphical tools, which enable the developer to carry out all required tasks for the design and creation of MAS, without any effort for writing even a single line of source code. In particular, AF comprises the Ontology Design Tool, the Behaviour 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 MAS is the design of one or more ontologies, in order for the agents to interoperate adequately. AF provides an Ontology Design Tool, which helps developers adopt ontologies defined with the Protégé-2000. The ontology files that are created with Protégé-2000 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.

By sharing ontology, agents become aware of common knowledge and therefore able to communicate and perform reasoning over common concepts. The AA agents are armed with a set of basic ontologies, which enable efficient agent communication in the AA environment. The users create their own ontologies, thus explicitly defining the semantics of new agent communication messages and agent activities. However, in this case, the users should also provide new behaviours (using the Behaviour Type Design tool) to enable the agents handle the messages. The ontologies need to be converted to appropriate JADE classes to become understandable by the agents. Therefore, the agents are not limited to share a specific set of ontologies. However, this ability can only be enabled in a manual fashion.

Once the agents are created, users can extend the existing ontologies, or create new ones in order to introduce new concepts. This operation may be performed using any ontology authoring tool, as long as it adheres to the ontology language standards, which are compatible with the Protégé tool, as e.g. Resource Description Framework (RDF) or OWL. The ontologies are finally imported into the AA environment using the Protégé compatibility function.

2.2.2 Creating behaviour types

The Behaviour Type Design Tool assists the developer in defining generic behaviour templates. Agent behaviours are modelled 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 modelled 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 customised 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 MAS for existing distributed systems

4 Java Expert System Shell (JESS; Friedman-Hill, 2003) 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 behaviour types, the Agent Type Definition Tool is provided to create new agent types that will be later used 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 MAS.

During the MAS instantiation phase, which is performed by 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 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 and the agent instances can be initialised. AF 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 knowledge models.

3 Agent training

As discussed already, the initial effort for the implementation of AA was motivated by the lack of an agent-oriented Software engineering tool coupled with Artificial Intelligence (AI) aspects. The ability to incorporate business knowledge into an agent's decision-making process is arguably essential for effective performance in dynamic environments and unfortunately, agent-oriented software engineering methodologies do not deal with agent reasoning issues (Witten and Frank, 2000). Moreover, building 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 the agent architecture, in order to be considered complete and generic.

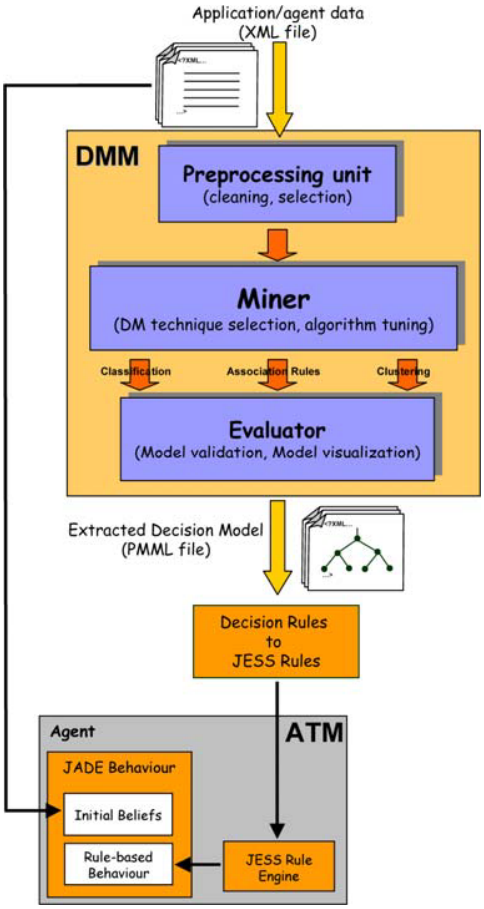
The knowledge extraction and incorporation processes are depicted in Figure 5. At first, let us consider an available source of data formatted in XML. The DMM receives data from the XML document and executes certain DM algorithms (suitable for generating a decision model), determined by the agent-application developer. The output of the DM procedure is formatted as a Predictive Model Markup Language (PMML) specifications document (The Data Mining Group, 2001).

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. PMML provides compatibility to other major DM software vendors, such as Oracle, SAS, SPSS, and MineIt.

The PMML document represents a knowledge model that expresses the referencing mechanism of the agent that 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, a robust mechanism for executing rule-based reasoning.

As shown in Figure 5, the DMM receives the application data in XML format and then performs data preparation in the preprocessing unit. The clean data are manipulated by the Miner, which applies DM algorithms and extracts the decision model after validating it in the Evaluator unit. The decision model in the form of the JESS rule engine finally becomes a part of the agent behaviour in the ATM module. In the context of training, new appropriate ontologies must be authored. These will provide an agent-readable description of knowledge pertaining to the decision models generated by the DM procedure.

Figure 5 Diagram of the agent training procedure

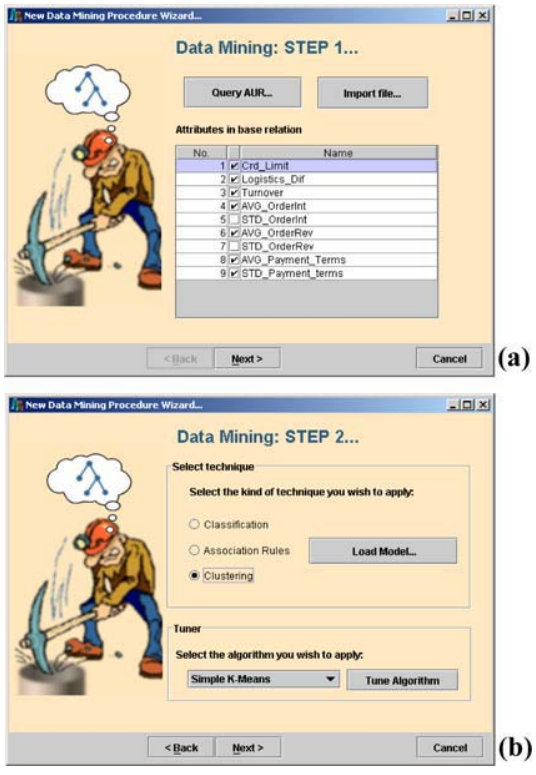


3.1 Mining background data for creating decision models

The mechanism for extracting knowledge from available data, in order to provide agents with reasoning, is based on the application of DM on background application-specific data (Symeonidis, Mitkas and Kehagias, 2001). From our experience with the application of AA to a number of industrial scenarios (Symeonidis, Kehagias and Mitkas, 2003; Symeonidis et al., 2003; Athanasiadis and Mitkas, 2004), we have ascertained that the enterprises' IT infrastructures generate and manipulate large amounts of data on a permanent basis, thus becoming suitable data providers for the needs of DMM.

In the initial phase of the DM procedure, the developer launches the GUI-based wizard depicted in Figure 6(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 6(b), the developer selects the type of the DM technique from a set of available options. In addition to some new algorithms (Athanasiadis et al., 2003), the DMM has incorporated a set of DM methods based on the Waikato Environment for Knowledge Analysis (WEKA) library and tools (Witten and Frank, 2000), appropriately extended to support PMML.

Figure 6 The two first steps of the DMM wizard



3.2 Embedding intelligence into agents

The completion of the training process requires the translation of the decision model extracted by DM 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.

4 Developed test cases

The AA consortium has developed three test cases, in order to prove the AA hypothesis and evaluate the platform usability and added value. The three test cases were deployed in the following domains:

1 in-house supply chain and Customer Order Management

2 real-time environmental monitoring and assessment

3 web-based Distributed Service Management (DSM).

The first test case addresses issues concerning Supply Chain Management, Customer Relationship Management, and Supplier Relationship Management. It has been implemented as MAS add-on to legacy Enterprise Resource Planning (ERP) systems, in order to provide intelligent policy recommendations on customer, supplier and inventory servicing (Athanasiadis et al., 2003; Symeonidis, Kehagias and Mitkas, 2003; Kehagias et al., 2004).

The second test case is an agent-based intelligent environmental monitoring system developed for assessing ambient air-quality. O_3RTAA deploys a community of software agents to monitor and validate measurements coming from several sensors, to assess air-quality, and finally, to fire alarms to appropriate recipients, when needed. DM techniques have been used for adding data-driven, customised intelligence into agents with successful results (Athanasiadis et al., 2003; Athanasiadis and Mitkas, 2004).

Finally, the third test case scenario addresses issues concerning web-based DSM. The MAS deployed is responsible for managing the physical (maintenance) services carried out at the customer's location, during the after sales phase. A number of agents, equipped with intelligence via DM, are in charge of process monitoring, decision-making, proactive event prediction, and customer segmentation (Agent Academy Consortium, 2000b).

5 Conclusions and future work

In this paper, we have pinpointed the inherent correlation between CE and agent technology. We presented AA, a generic multi-agent development framework for

constructing MAS, or single agents. We argued that existing tools and infrastructures for agent development provide suboptimal solutions in the case of designing and implementing CE processes by the use of MAS. In contrast, the AA 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. This architecture proves capable of grasping and reproducing the core prerequisites of the CE design processes. We have presented the functional architecture of our framework; we shortly demonstrated a representative scenario for deploying MAS, discussed the details of the agent 'training' process, and briefly mentioned three, already developed fully functional test case scenarios.

Through our experience with AA, we are convinced that this development environment significantly reduces the programming effort for building CE processes, both in terms of time and code efficiency. The developed test cases have indicated that AA meets its initial objective for building agent-based applications 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 dependent upon the amount of available data.

The AA framework is the result of a development effort, which begun 3 years ago. Currently, the second stable version (1.2) of the platform is publicly available on SourceForge (Agent Academy on sourceforge, 2003) as an open source product. Our near future work involves the exhaustive testing of the platform and thorough exploration of scheduling, coordination, and collaboration issues, which are of great importance in the area of CE.

References

Agha, G. (1986) *ACTORS: A Model of Concurrent Computation in Distributed Systems.* Cambridge, MA: The MIT Press.

Agha, G. and Hewitt, C. (1988) 'Concurrent programming using actors', in Y. Yonezawa and M. Tokoro (Eds). *Object-Oriented Concurrent Programming* (pp.37–57). Cambridge, MA: MIT Press.

Agha, G., Wegner, P. and Yonezawa, A. (Eds.) (1988) *Research Directions in Concurrent Object-Oriented Programming*. Cambridge, MA: The MIT Press.

Agent Academy Consortium (2000) *Technical Specifications of the Three Test Cases*, Available at: http://AgentAcademy.iti.gr/downloads.htm.

Agent Academy on sourceforge (2003), Available at: https://sourceforge.net/projects/agentacademy.

Athanasiadis, I.N. and Mitkas, P.A. (2004) 'An agent-based intelligent environmental monitoring system', *Management of Environmental Quality: An International Journal*, Vol. 15, pp.238–249.

Athanasiadis, I.N., Kaburlasos, V.G., Mitkas, P.A. and Petridis, V. (2003) 'Applying machine learning techniques on air quality data for real-time decision support', Paper presented in the Proceedings of the *First International NAISO Symposium on Information Technologies in Environmental Engineering (ITEE'2003)*, *Gdansk* (p.51). Poland: Academic Press.

Cutkosky, M.R., Engelmore, R.S., Fikes, R.E., Genesereth, M.R., Gruber, T.R., Mark, W.S., Tenebaum, J.M. and Weber, J.C. (1993) 'PACT: an experiment in integrating concurrent engineering systems', *IEEE Computer*, Vol. 26, pp.28–37.

FIPA, Foundation for Intelligent Physical Agents Specifications (2000) 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/.

Gutknecht, O. and Ferber, J. (2000) 'Madkit: a generic multi-agent platform', *The 4th International Conference on Autonomous Agents, Barcelona, Spain*.

Grosso, W.E., Eriksson, H., Ferguson, R.W., Gennari, J.H., Tu, S.W. and Musen, M.A. (1999) 'Knowledge modeling at the millennium', *The Design and Evolution of Protege-2000*, Available at: http://protege.stanford.edu.

Jeon, H., Petrie, C. and Cutkosky, M. (2000) 'JATLite: a Java agent infrastructure with message routing', *IEEE Internet Computing*, Vol. 4, pp.87–96.

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

Kehagias, D., Chatzidimitriou, K., Symeonidis, A.L. and Mitkas, P.A. (2004) 'Information agents cooperating with heterogeneous data sources for customer-order management', *Special Track on Agents, Interactions, Mobility, and Systems (AIMS), held at the 19th ACM Symposium on Applied Computing (SAC 2004), Nicosia, Cyprus*.

McGuire, J., Huakka, D., Weber, J., Tenenbaum, J., Gruber, T. and Olsen, G. (1993) 'SHADE: technology for knowledge-based collaborative engineering', *Journal of Concurrent Engineering: Applications and Research*, Vol. 1, pp.137–146.

Mitkas, P.A., Kehagias, D., Symeonidis, A.L. and Athanasiadis, I.N. (2004) 'A framework for constructing multi-agent applications and training intelligent agents', in P. Giorgini, J.P. Mueller and J. Odell (Eds). *Agent Oriented Software Engineering IV* (pp.96–109). LNCS 2935, Springer-Verlag.

Nwana, H., Ndumu, D., Lee, L. and Collis, J. (1999) 'ZEUS: a tool-kit for building distributed multi-agent systems', *Applied Artifical Intelligence Journal*, Vol. 13, pp.129–186.

Petrie, C., Cutkosky, M., Webster, T., Conru, A. and Park, H. (1994) 'Next link: an experiment in coordination of distributed agents', *The AID '94 Workshop on Conflict Resolution, Lausanne*.

Shen, W., Norrie, D.H. and Barthes, J.P.A. (2001) *Multi-agent systems for concurrent intelligent design and manufacturing* (Taylor and Francis, Great Britain).

Suguri, H., Kodama, E., Miyazaki, M., Nunokawa, H. and Noguchi, S. (2001) 'Implementation of FIPA ontology service', Paper presented in the Proceedings of the *Workshop on Ontologies in Agent Systems*, 5th International Conference on Autonomous Agents Montreal, Canada.

Symeonidis, A.L., Mitkas, P.A. and Kehagias, D. (2002) 'Mining patterns and rules for improving agent intelligence through an integrated multi-agent platform', *The 6th IASTED International Conference, Artificial Intelligence and Soft Computing ASC*, Banff, Alberta, Canada.

Symeonidis, A.L., Kehagias, D. and Mitkas, P.A. (2003) 'Intelligent policy recommendations on enterprise resource planning by the use of agent technology and data mining techniques', *Journal of Expert Systems with Applications*, Vol. 25, pp.589–602.

Symeonidis, A.L., Kehagias, D., Koumpis, A. and Vontas, A. (2003) 'Open source supply chains', Paper presented in the Proceedings of the *10th International Conference on Concurrent Engineering (CE-2003)*, Vol. 1, pp.31–36.

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

Wooldridge, M. (1997) 'Agent-based software engineering', *IEEE Proceedings in Software Engineerin*g, Vol. 144, pp.26–37.

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