# Supporting Agent-Oriented Software Engineering for Data Mining Enhanced Agent Development

Andreas L. Symeonidis[1,2], Panagiotis Toulis[3], and Pericles A. Mitkas[1,2]

[1] Electrical & Computer Engineering Department,
Aristotle University of Thessaloniki
[2] Informatics and Telematics Institute, CERTH
Thessaloniki, Greece
asymeon@eng.auth.gr, mitkas@auth.gr
[3] Department of Statistics, Harvard University, Boston, MA, USA
ptoulis@fas.harvard.edu

**Abstract.** The emergence of Multi-Agent systems as a software paradigm that most suitably fits all types of problems and architectures is already experiencing significant revisions. A more consistent approach on agent programming, and the adoption of Software Engineering standards has indicated the pros and cons of Agent Technology and has limited the scope of the, once considered, programming 'panacea'. Nowadays, the most active area of agent development is by far that of intelligent agent systems, where learning, adaptation, and knowledge extraction are at the core of the related research effort. Discussing knowledge extraction, data mining, once infamous for its application on bank processing and intelligence agencies, has become an unmatched enabling technology for intelligent systems. Naturally enough, a fruitful synergy of the aforementioned technologies has already been proposed that would combine the benefits of both worlds and would offer computer scientists with new tools in their effort to build more sophisticated software systems. Current work discusses *Agent Academy*, an agent toolkit that supports: a) rapid agent application development and, b) dynamic incorporation of knowledge extracted by the use of data mining techniques into agent behaviors in an as much untroubled manner as possible.

## 1 Introduction

More than a decade ago, agents appeared as a 'hype' that was abstract enough to fit any given or future problem. It is only recently that their range of applicability has been narrowed down to specific application domains (e.g. Grid computing [7], electronic auctions [8], autonomic computing [11] and social networks [9]) that exploit the beneficial characteristics of agents. Still, software practitioners are reluctant in incorporating agent solutions to solve real-world problems, even in the case of the above-mentioned domains, where agents have proven to be efficient. This reluctance has been attributed to many reasons that range from

L. Cao et al.: ADMI 2012, LNAI 7607, pp. 7–21, 2013.
© Springer-Verlag Berlin Heidelberg 2013

the lack of consensus in definitions and the interdisciplinary nature of agent computing to the lack of tools and technologies that truly allege the real benefits of *Agent Technology* (AT) [12]. In fact, AT still seems like it is missing its true scope.

May one take a closer look at the domains that AT is considered as the proper programming 'metaphor', one may notice that, apart from their dynamic nature and their versatility, these domains have another thing in common: they generate vast quantities of data at extreme rates. Data from heterogeneous sources, of varying context and of different semantics become available, dictating the exploitation of this "pile" of information, in order to become useful bits and pieces of knowledge, the so-called *knowledge nuggets* [13]. Nuggets that will be used by software systems and will add intelligence, in the sense of adaptability, trend identification and prevention of decision deadlocks.

To this end, *Data Mining* (DM) appears to be a suitable paradigm for extracting useful knowledge. The application domain of Data Mining and its related techniques and technologies have been greatly expanded in the last few years. The development of automated data collection tools has fueled the imperative need for better interpretation and exploitation of massive data volumes. The continuous improvement of hardware, along with the existence of supporting algorithms has enabled the development and flourishing of sophisticated DM methodologies. Issues concerning data normalization, algorithm complexity and scalability, result validation and comprehension have been successfully dealt with [1,10,15]. Numerous approaches have been adopted for the realization of autonomous and versatile DM tools to support all the appropriate pre- and post-processing steps of the knowledge discovery process in databases [5, 6].

From all the above, the synergy of AT with DM seems promising towards providing a thrust in the development and establishment of intelligent agent systems [3, 4]. Knowledge hidden in voluminous data repositories can be extracted by data mining, and provide the inference mechanisms or simply the behavior of agents and multi-agent systems. In other words, these knowledge nuggets may constitute the building blocks of agent intelligence. We argue that the two, otherwise diverse, technologies of data mining and intelligent agents can complement and benefit from each other, yielding more efficient solutions.

However, while the pieces are already there, the puzzle is far from complete. No existing tool provides solutions to the theoretical problems of researchers striving to seamlessly integrate data mining with agent technology, nor to the practical issues developers face when attempting to build even simple multi-agent systems (MAS).

*Agent Academy* (AA) is a practical approach to the problem of establishing a working synergy between software agents and data mining. As a lower CASE (Computer Aided Software Engineering) tool, Agent Academy combines the power of two proven and robust software packages (APIs), namely JADE [2] and WEKA [15], and along with the related theoretical framework, AA sketches a new methodology for building data mining-enabled agents. As a development tool, Agent Academy is an open source IDE with features like structured code

editing, graphical debugging and beginner-friendly interface to data mining procedures. Finally, as a framework, Agent Academy is a complete solution enabling the creation of software agents at any level of granularity, with specific interest on intelligent systems and decision making.

Technically, Agent Academy acts as a hub between the agent development phase and the data mining process by adding models as separate and independent threads into the agent's thread pool. These execution units, usually called behaviors in software agent terminology, are reprogrammable, reusable and even movable among agents. Using Agent Academy, intelligent agents are not an afterthought, but rather the very basic capability that this engineering tool has to offer.

The remainder of this chapter is organized as follows: Section 2 presents the methodology to follow in order to ensure the synergy between AT and DM. Section 3 discusses Agent Academy in detail, the development framework for DM-enriched MAS, while Section 4 summarizes work presented and discusses future extensions.

## 2 Integrating Agents and Data Mining

The need to couple AT with DM comes from the emergent need to improve agent systems with knowledge derived from DM, so as to strengthen their existence in the software programming scenery. Nevertheless, coupling of the two technologies does not come seamlessly, since the inductive nature of data mining imposes logic limitations and hinders the application of the extracted knowledge on deductive systems, such as multi-agent systems. The methodology and the supporting tool described within the context of this paper take all the relevant limitations and considerations into account and provide a pathway for employing data mining techniques in order to augment agent intelligence.

Knowledge extraction capabilities must be present in agent design, as early as in the agent modeling phase. During this process, the extracted models of the DM techniques applied become part of the knowledge model of agents, providing them with the ability to enjoy DM advantages.

In Symeonidis and Mitkas [14], the reader may find more details on the unified methodology for transferring DM extracted knowledge into newly created agents. Knowledge diffusion is instantiated in three different ways, always with respect with the levels of diffusion of the extracted knowledge (DM on the application level of a MAS, DM on the behavioral level of a MAS, DM on the evolutionary agent communities). The iterative process of retraining through DM on newly acquired data is employed, in order to enhance the efficiency of intelligent agent behavior.

As already mentioned, the methodology presented is also supported by the respective toolkit. Using Agent Academy, the developer may automate (or semi-automate) several of the tasks involved in the development and instantiation of a MAS. He/she follows the steps shown in Figure 1, in order to build a DM-enhanced MAS in an untroubled manner. Details on the Agent Academy framework as discussed next.

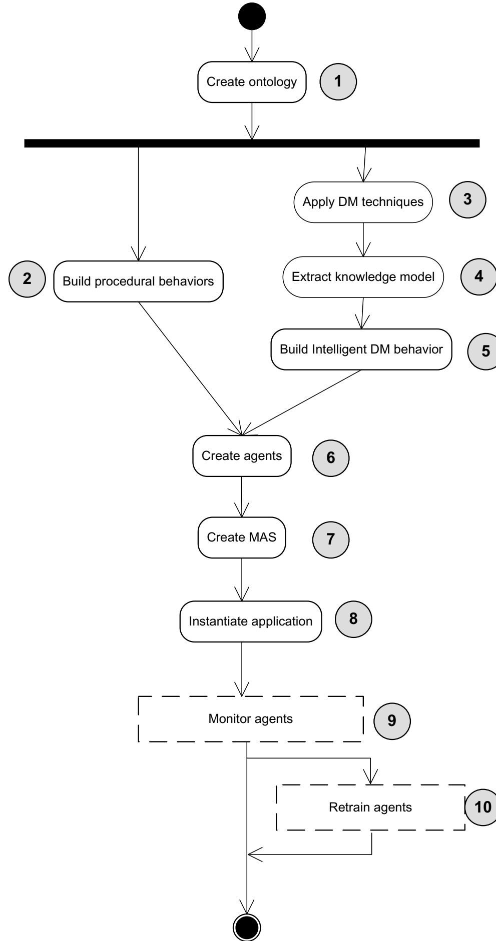


**Fig. 1.** The MAS development steps

## 3 Agent Academy

### 3.1 Introduction

Agent Academy is an open-source framework and integrated development environment (IDE) for creating software agents and multi-agent systems, and for augmenting agent intelligence through data mining. It follows the aforementioned methodology, in order to support the seamless integration of Agent Technology and Data Mining.

The core objectives, among others, of Agent Academy are to:

- Provide an easy-to-use tool for building agents, multi-agent systems and agent communities
- Exploit Data Mining techniques for dynamically improving the behavior of agents and the decision-making process in multi-agent systems

- Serve as a benchmark for the systematic study of agent intelligence generated by training them on available information and retraining them whenever needed.
- Empower enterprize agent solutions, by improving the quality of provided services.

*Agent Academy has been implemented upon the JADE and WEKA APIs (Application Programming Interfaces), in order to provide the functionality it promises. The initial implementation of AA was funded under the fifth Framework Program, where the theoretical background was formulated. After the successful completion of the project the second version of Agent Academy (AA-II) was built, where emphasis was given on the user interface and functionality. Agent Academy is built in Java and is available at Sourceforge (http://sourceforge.net/projects/agentacademy). The current release contains 237 Java source files and is spun on over 28,000 lines of code.*

The original Agent Academy vision stands on the edge of being an intelligent agent research tool. Nevertheless, considerable effort was given in order to provide adequate quality level and user-friendliness, so as to support industrial-scope agent applications. To this end, one of the pivotal requirements for AA was to provide maximum functionality with the minimum learning curve. Agent Academy has been built around well-known concepts and practices, not trying to introduce new APIs and standards, rather to increase applicability of the already established ones. The novelty of AA lies on the fruitful integration of agent and data mining technologies, and the methodology for doing so. In fact, Agent Academy proposes a new line of actions for creating DM-enhanced agents, through a simple and well-defined workflow. Prior to unfolding the details of this workflow we define the concepts and conventions AA is founded on (Table 1).

In a typical scenario the developer should follow the methodology described in the previous section in order to build a new agent application. He/she should first create behaviors, as orthogonal as possible. while in a parallel process, he/she should build the data mining models which apply to the application under development. Next, everything should be organized/assigned to agents, in order to finally build the multi-agent system by connecting the developed software agents.

### 3.2 Agent Academy Architecture

The AA IDE offers a complete set of modules that can guide and help through the entire process. These modules, namely the *Behavior Design Tool*, the *Agent Design Tool*, and the *MAS Design Tool* support the respective development phases of an Agent Academy project through graphical user interfaces, while they also support code editing, debugging and compilation.

Nevertheless, AA follows a component-based approach, where each of the modules is loosely coupled with the others. This way the user may decide to

build (JADE compliant) behaviors or (WEKA) data mining models with any other tool/framework and just import them in AA, just by providing the relative path of their location. In the following sections we first provide a qualitative description of the three core modules, and then continue with a more technical description on them.

**Table 1.** Agent Academy conventions

| | |
|---|---|
| Behaviors | Blocks of code sharing the same functional description, which can, conceptually, be grouped together. Based on the JADE paradigm, behaviors are essentially distinct threads of execution that run in parallel and are assigned to agents, in order to exhibit desired (agent) properties |
| Intelligent DM behaviors | Technically, the combination of plain (procedural) agent behaviors with data mining models. Any possible type of data mining generated knowledge can serve as the element of an intelligent behavior, e.g. classification intelligent behavior, clustering intelligent behavior, etc. Such types of behaviors are based on data mining models built by the use of the WEKA API (most of the times offline). AA then uptakes the task of transforming the WEKA-generated model into a compiled JADE behavior which can, after that point, operate indistinguishably in any JADE-developed MAS. This type of behavior is a special case of an Intelligent behavior |
| Agents | Distinct software entities, programming metaphors, embracing a groups of behaviors. Agents in AA can communicate with others through well established protocols (the FIPA Agent Communication Language - ACL) |
| Intelligent DM Agents | Agents that have at least one intelligent DM behavior in their collection of behaviors. In practice, such agents can emulate any kind of high-level cognitive task backed by the corresponding DM model (classify, cluster, associate etc). This type of agents is a special case of an intelligent agent |
| Multi-agent Systems | Collections of agents, intelligent or not, operating under the same environment |

**Agent Academy Overview.** Agent Academy organizes work into projects. Each project has a private space that contains all the necessary elements code for the development and instantiation of an intelligent MAS. These elements may be of two types: i) software snippets, i.e. programming code resulting to behavior classes and agent classes, and ii) data (either in the form of a file or as database connection), where data mining will be applied on in order to generate DM models.

**Creating Agent Behaviors.** The notion of agent behavior is very popular among agent-oriented software methodologies and represents essentially a block of code that encapsulates an execution thread, aiming to fulfill one, or a few well-defined tasks. Specifically in the JADE API, which is employed by Agent Academy, an agent behavior is realized as a distinct Java class.

The *Behavior Design Tool* (BDT) of Agent Academy implements a minimal code editing tool with code automation functionality. BDT comprises two panes: the BDT toolbar and the code editing area. Although minimal, BDT should be considered as a fully functional agent behavior creation tool, since it provides features such as automatic code generation for typical agent code blocks (such sending/receiving FIPA ACL messages), structured code tools, text editing and compilation options. Apart from plain (procedural) behaviors, Agent Academy defines another category of agent behaviors, namely the *Intelligent DM Behaviors*, which encapsulate DM models in an AA-compliant code wrapper. These "intelligent" pieces of code are created through a 5-step process, which employs the WEKA API to build the data mining models and then compile them into JADE-compliant behaviors.

**Creating Software Agents.** The *Agent Design Tool* (ADT) is the functional equivalent of BDT for agent creation. The definition of "agency" within the development context of Agent Academy is absolutely generic: an agent is simply thought of as a meaningful grouping of agent behaviors, aiming to a specific set of tasks, following a specific workflow. Thus, in order to build an agent in AA, the developer needs only to assign the agent with a set of behaviors and a sequence of behavior execution. All project behaviors, both *plain* and *intelligent*, are available for insertion to the agents' execution pool, through the use of automatic IDE tools.

ADT comprises the code editing area and the ADT toolbar that provides code generation and debugging functions. ADT can be used to assign behaviors to agents, compile the generated code, generate agent class files, and debug agent execution.

**Instantiating Multi-agent Systems.** The *MAS Design Tool* (MDT) allows the developer to organize agents into multi-agent systems. It provides the tools to initialize, pause and terminate the MAS, as well as to monitor agent state (alive, dead, etc). Typically, this step concludes an Agent Academy project, i.e. the development and instantiation phase of a MAS.

### 3.3 Technical Details of Agent Academy

Upon project creation, Agent Academy creates a specific folder structure under the directory:

```
<INSTALL_DIR>/user/projects/<PROJECT_NAME>
```

In the previous section we provided an overview of the AA modules. Here, we further elaborate on them.

**The Behavior Design Tool.** The first step in the Agent Academy workflow is the creation of the agent behaviors which are the software blocks that encapsulate

the functionality of the agents. These behaviors are typical Java classes and can be created using the Behavior Design Tool.

BDT provides a set of features that may assist in the creation of the agent behaviors. AA behaviors are extensions of well-defined JADE behaviors; this implies that developers already familiar with the JADE framework may use BDT to build behaviors directly. Even in the case of unexperienced developers, though, AA provides automated code generators to help the user through the behavior creation process. The workflow of creating a new behavior entails the following:

1. Initializing the behavior, setting names and identifiers, and defining the type of behavior
2. Adding fields, methods and blocks of code, through the *Structured Edit Action* - SEA module, a specific-purpose module of the Agent Academy IDE. The developer may, of course, write the communication source code himself/herself, in case he/she prefers
3. Specifying FIPA-compliant agent communication, through the respective messaging buttons located on the BDT toolbar, which automatically generate code blocks for sending and receiving ACL messages. The developer may, of course, write the communication source code himself/herself, in case he/she prefers
4. Optionally documenting the behavior, by providing a small description and saving it to the project behaviors *notepad*
5. Saving the newly created source code and compiling, directly from the AA environment.

This workflow is supported by the functions depicted in Figure 2.

Upon behavior initialization, the developer is requested to define the name of the behavior, the sub-package the behavior should belong and its type. All the fundamental JADE behaviors types are supported within the context of AA:

- `OneShotBehavior`, where a behavior is executed only once
- `CyclicBehavior`, where a behavior is executed ad infinitum
- `TickerBehavior`, where a behavior is executed periodically, with respect to a user-defined time slot
- `WakerBehavior`, where a behavior is executed when certain conditions are met.

In addition, the JADE framework provides the archetypical behavior class (`SimpleBehavior`), which can be modified in order to produce any kind of functionality inside the generated behavior.

As already mentioned, the AA IDE supports automated code authoring for generating FIPA-compliant ACL messages. This kind of functionality, along with the documentation options AA provides, allow for **rapid agent-based application development**.

One of the most interesting features of the BDT is the SEA, which offers a systematic way to create Java class members along with a visual ‘summary’ of the behavior functionality. Through the SEA feature developers may:

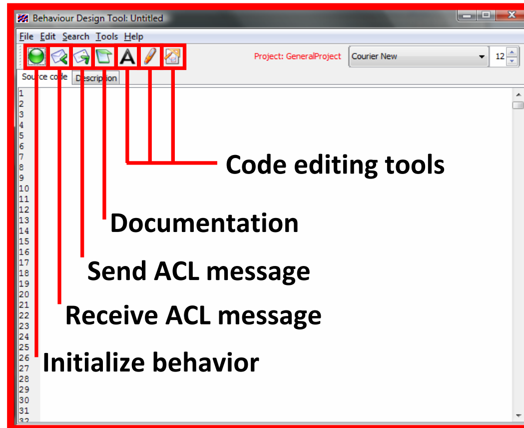


**Fig. 2.** Functionality provided for creating agent behaviors

- Define class/instance fields and automatically generate `get/set` methods for them (much like the Properties of the C# language)
- Define methods by providing their signatures
- Create documented code blocks.

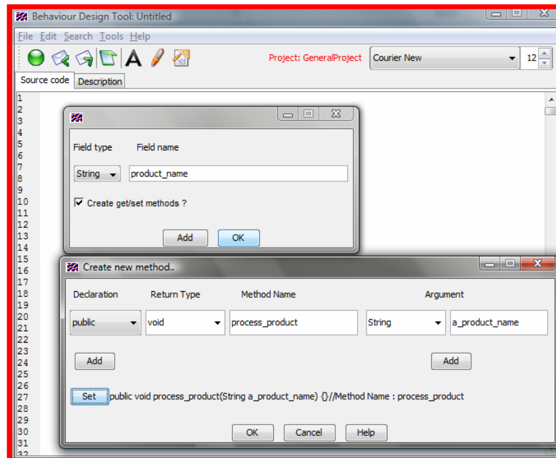


**Fig. 3.** Creating fields and methods

Developers have the ability to transfer code blocks among methods and also dynamically edit their content. The *Description* tab of the source code editor can be used to provide with a brief summary of what the behavior really does. As long as the developer conforms to the AA code structuring standards, this approach can be quite beneficial for code clarity and maintainability.

Figure 3 illustrates the use of SEA in defining a class field named `product_name` of type `String`, creating the `get/set` methods for it and

then defining a new method named `process_product()`. May one use the code editor, one can write the fundamental code blocks of the behavior, which can be embedded to any method defined in the previous steps. Optionally they can be tagged with a short text which describes their functionality. Figure 4 depicts the process of creating a new source code block, tagged as "*PROCESS THE PRODUCT*", and then adding the source code to implement the desired functionality. By using the *Method* process list, one can embed the newly created code block to any of the available methods inside the behavior. In this case Agent Academy will automatically produce the resulting source code, but will also inject some additional AA-oriented code useful for debugging purposes. In fact, the additional code denotes the start and exit points of every behavior created with the Behavior Design Tool, and by using a special debugging component of the Agent Design Tool, it is easy monitor which behavior is executed at any given time. Finally, the tags that are being assigned to each code block are listed in succession at the *Description* tab, which summarizes quickly and efficiently the behavior functionality.

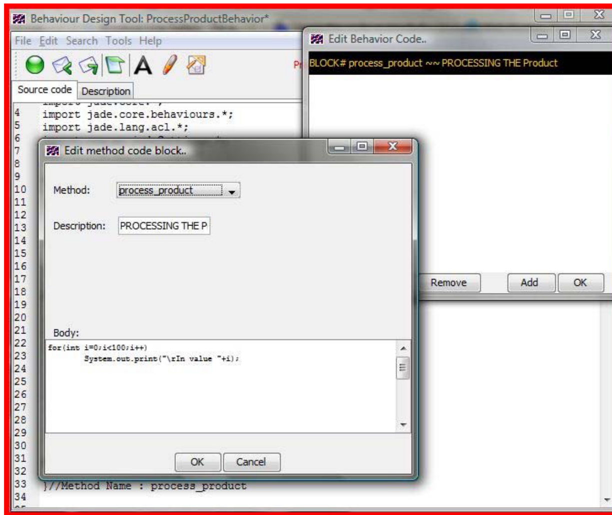


**Fig. 4.** Creating code blocks

It should be denoted that all source code libraries needed both for performing JADE-related tasks and Agent Academy structures are automatically imported during behavior initialization.

**Intelligent Data Mining Behaviors.** As already discussed, an Intelligent DM behavior in Agent Academy is a typical JADE behavior that also incorporates knowledge extracted by the application of DM techniques on problem related data. This knowledge is represented by a DM model, like the ones that are generated by the WEKA API. When building such an intelligent behavior, the

user typically follows all the steps of the KDD process (section 2). Through the AA-adapted WEKA GUI, he/she supplies the dataset DM will be applied on, performs preprocessing, feature selection, algorithm parametrization, and model evaluation, like in any given DM problem. After the DM model has been built, the developer has to either approve it or request retraining, which implies reconfiguration of the KDD process at any point (preprocessing, algorithm parameters, etc). Upon model approval, the developer calls Agent Academy to embed the generated model in a JADE behavior and compile it.

Following the AA conventions in the development of a AA project, data to be mined are stored in a specific location:

```
<INSTALL_DIR>/user/projects/<PROJECT_NAME>/data/DMRepository/
```

To initiate the DM behavior building process, the user has to navigate from "*Tools*" to "*Create a Data Mining Behavior*", from the initial screen of Agent Academy. In order to wrap the DM model in a JADE-compliant schema, AA creates a JADE `CyclicBehavior`, configured to wait for ACL messages adhering to the following two constraints:

- The Ontology of the message should be set to the constant `AAOntoloty.` *ONTOLOGY_NAME*
- The ACL message protocol should be set according to the specific operation that takes place. For example, if the DM model performs classification, the protocol should be set to `AAOntology.CLASSIFY_REQUEST`

These two constraints guarantee that an agent with the DM behavior will succeed in communicating with other agents on the specific topic (e.g. classification task). In addition, the actual message content should be an object of the specific type class that extends the `org.aa.ontology.`*DM_TECHNIQUE_REQUEST* class (in case of classification this should be the `org.aa.ontology.ClassifyRequest` class). This object, in fact, wraps the actual data tuple the model will be applied on.

In response, the agent executing the intelligent behavior shall generate an ACL message with the protocol field set to `AAOntology.CLASSIFY_RESPONSE` (in the case of classification) and the actual content of the message will be a serialized object, namely of type `org.aa.ontology.ClassifyResponse`. This object, in fact, wraps the output of the DM model, with respect to the input data tuple the sender provided.

The actual mechanism of building an intelligent DM behavior is a bit more complicated. When the developer builds such a behavior after having produced the respective DM model, AA produces three additional files with extensions *.aadm*, *.aainst* and *.dat*. These files contain the training dataset the model was built on, along with the generated model itself. When an operation, such as a classification,, is requested, Agent Academy retrieves the "*.aainst*" file and regenerates it as a functional Java class, which is able to carry out the data mining operation. This way, both the model and the agent behavior can be

created on-the-fly. The requests and responses that agents exchange in this context should be subclasses of the *Request* and *Response* classes defined in the `org.aa.ontology` package. AA by default defines such classes, like `ClassifyRequest` and `ClusterRequest`, nevertheless the user is allowed to define more complex structures. These objects practically carry the data for processing in a field of type `org.aa.tools.DMList`, which is essentially a vector of values, *double* or *String* type.

### 3.4 Agent Design Tool

Agent Academy treats agents as logical groupings of software behaviors, based on the idea that an agent is completely defined by the way it actually acts. Thus, ADT provides all the functionality for inserting and organizing behaviors in an agent execution pool. Figure 5 illustrates the features of the Agent Design Tool, listed below:

1. **Agent definition**, during which the user sets the name of the agent and the package to which it belongs
2. **Behavior assignment**, which browses for project behaviors and lists them for the user to assign. This feature is available through the "*Add Behavior*" option.
3. **Text editing and compilation options**, where the user may manually edit the source code, in case he/she deems necessary. Through this option the developer may also invoke the Java compiler, so that the new agent class is generated and is mapped to the agent
4. **Agent Monitoring**, which provides a graphical tool for validating agent execution
5. **Intelligent DM Behavior assignment**, which enables the user to assign intelligent DM behaviors to the agent behavior pool. Through this feature, the user may also request for model retraining of a specific DM Behavior, in case of poor performance or existence of update data. Retraining may be performed on-the-fly, since all the model parameters are available and are loaded on behavior initialization at runtime.

In order to initialize an agent, the developer has to select a name and optionally select the package to which it shall be assigned to. Agent Academy considers a default package for every behavior or agent, and any user-defined package should be a sub-package of the default one. Thus, if `<PROJECT_NAME>` denotes the project name, behaviors are compiled by default as classes and are stored in the `<PROJECT_NAME>.behaviors` package, while agents are compiled and stored in the `<PROJECT_NAME>.agents` package. Naturally, the user is allowed to organize class files into more packages, nevertheless it is mandatory all of them to be sub-packages of the default packages. When the user browses for behaviors to add to the agent, a list of Java classes are available through a regular 'Open File Dialog'. After the behaviors have been added, Agent Academy automatically generates all the necessary code, so there is usually no need for extra work. However there are two points that the user should take care of:

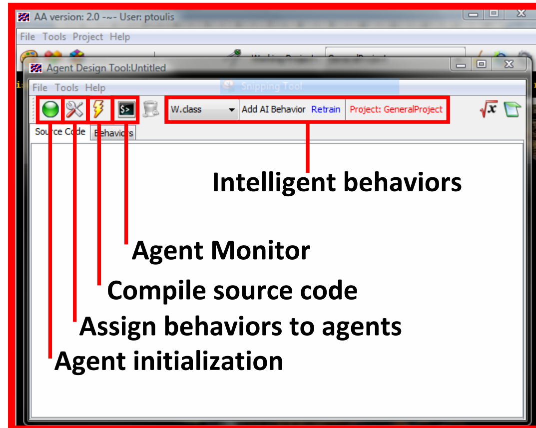


**Fig. 5.** Agent Design Tool features

1. Up to this point, there is no way for Agent Academy to infer the ontology the agents should communicate on. Therefore, the following source code line is added in the agent source code:

```
this.getContentManager().registerOntology(**);
```

   The user, then, should replace the '**' with the desired ontology for the particular application. If a user-defined ontology does not exist, this line can simply be commented out.
2. Special attention should be given to the way AA handles behavior constructors. Currently, reflection is not used on behavior Java classes; as a result, only constructors with no arguments are assumed and at any other case users should manually edit the generated source code.

Given that all these issues are resolved, the generated source code can be compiled without problems.

A very important feature of ADT is the *Agent Monitor*, which can help greatly with the runtime debugging of the agent application under development. It is a graphical tool that enables AA developers to monitor when and in what order the behaviors of a specific agent are executed. Given that at the core of the agent-based software philosophy lies the multi-threaded (or parallel) programming model, this feature guarantees that agents will perform exactly as they were designed to. Additionally, the *Agent Monitor* provides with information on the message queue of agents, enabling the developer to inspect how messages are sent, received, and processed.

Figure 6 provides a screenshot of the *Agent Monitor* launched on an agent with three distinct behaviors. At any given time, only one behavior should be active, marked with a green square as shown in the figure. Using the monitor, the user may also inspect how behaviors are interchanged during execution, as well as the status and the message queue of the agent.

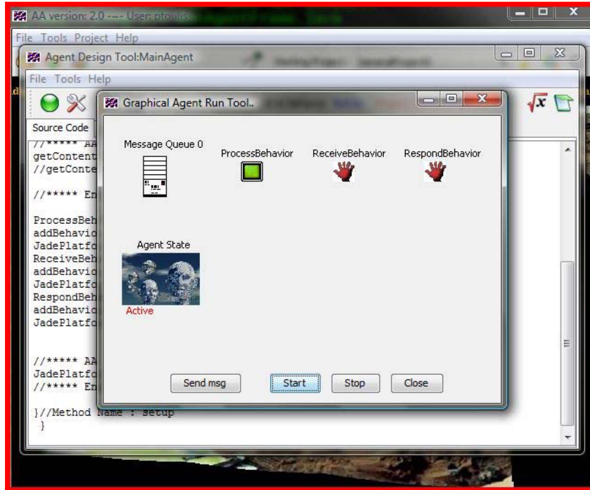


**Fig. 6.** The Agent Monitor

Another important feature of ADT is the intelligent DM behaviors toolbar. All available DM behaviors can be browsed in a drop down list and any of them can be added to the agent behavior pool with a simple click of a button. AA provides access to all intelligent DM behaviors that belong to the active project, so that it is possible for two or more agents to share them. When a DM behavior is retrained, it is dynamically updated to all agents employing it. And, though, agents of the same project may share behaviors that belong to that project, at the time being it is not possible for two projects to share code.

**Multi-Agent System Design Tool.** Having created all behaviors and all agents that implement the business logic of the application, the final step is to group everything under a Multi-Agent System. This is succeeded through the MAS design tool. AA saves all the information of the agents participating in the MAS in a simple text file with the extension *.aamas* (abbreviation for "Agent Academy MAS"), for latter use. In addition to defining the MAS, MDT initializes a JADE platform instance and allows the user to start/pause/stop the execution of the MAS. Parallel to the instantiation of the MAS, AA also loads the JADE sniffer, a tool for monitoring messages exchanged among agents.

## 4 Summary and Future Work

Within the context of this paper we have presented a methodology that provides the ability to *dynamically* embed DM-extracted knowledge to agents and multi-agent systems. Special emphasis is given on Agent Academy, the developed platform that supports the whole process of building DM-enhanced agent systems. It helps agent programmers to easily create agent behaviors, extract knowledge models based on Data Mining and integrate all these into fully working MAS.

AA can be thought of as a lower CASE (Computer Aided Software Engineering) tool, combining the power of two proven and robust software APIs, JADE and WEKA. Agent Academy is an open source IDE with features like structured code editing, graphical debugging and beginner-friendly interface to data mining procedures.

Future directions include the incorporation of the Agent Performance Evaluation (APE) [13] into the AA framework, in order to provide developers with the tools for evaluating the performance of the agent systems developed.

## References


1. Adriaans, P., Zantinge, D.: Data Mining. Addison-Wesley, Reading (1996)
2. Bellifemine, F., Poggi, A., Rimassa, G.: Developing Multi-agent Systems with JADE. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS (LNAI), vol. 1986, pp. 89–101. Springer, Heidelberg (2001)
3. Cao, L.: Data mining and multi-agent integration. Springer-Verlag New York Inc. (2009)
4. Cao, L., Weiss, G., Yu, P.S.: A brief introduction to agent mining. In: Autonomous Agents and Multi-Agent Systems (2012)
5. Chen, M.-S., Han, J., Yu, P.S.: Data mining: an overview from a database perspective. IEEE Trans. on Knowledge and Data Engineering 8, 866–883 (1996)
6. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P.: Knowledge discovery and data mining: Towards a unifying framework. In: Knowledge Discovery and Data Mining, pp. 82–88 (1996)
7. Foster, I., Jennings, N.R., Kesselman, C.: Brain meets brawn: why grid and agents need each other. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS 2004, pp. 8–15 (2004)
8. Gimpel, H., Jennings, N.R., Kersten, G., Okenfels, A., Weinhardt, C.: Negotiation, auctions and market engineering. Springer (2008)
9. Gruber, T.: Collective knowledge systems: Where the social web meets the semantic web. Web Semantics: Science, Services and Agents on the World Wide Web 6(1), 4–13 (2008); Semantic Web and Web 2.0
10. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers (2001)
11. McCann, J.A., Huebscher, M.C.: Evaluation Issues in Autonomic Computing. In: Jin, H., Pan, Y., Xiao, N., Sun, J. (eds.) GCC 2004. LNCS, vol. 3252, pp. 597–608. Springer, Heidelberg (2004)
12. Kitchenham, B.A.: Evaluating software engineering methods and tool, part 2: selecting an appropriate evaluation method technical criteria. SIGSOFT Softw. Eng. Notes 21(2), 11–15 (1996)
13. Maimon, O., Rokach, L. (eds.): Soft Computing for Knowledge Discovery and Data Mining. Springer (2008)
14. Symeonidis, A.L., Mitkas, P.A.: Agent Intelligence Through Data Mining. Springer Science and Business Media (2005)
15. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco (2005)