Drag-and-Drop Migration: An Example of Mapping User Actions to Agent Infrastructures

Silvan Kaiser
DAI-Labor, Technische Universität Berlin
Ernst-Reuter-Platz 7
10587 Berlin, Germany
silvan.kaiser@dai-labor.de

Michael Burkhardt
DAI-Labor, Technische Universität Berlin
Ernst-Reuter-Platz 7
10587 Berlin, Germany
michael.burkhardt@dai-labor.de

Jakob Tonn
DAI-Labor, Technische Universität Berlin
Ernst-Reuter-Platz 7
10587 Berlin, Germany
jakob.tonn@dai-labor.de

ABSTRACT

Runtime management of a distributed Multi-Agent System is a complex task. Tools that offer a generic solution for this problem and that are intuitive to use only exist in a very limited way so far. In this paper, we take the example of a Drag-And-Drop migration feature to show a concept and a prototype implementation for an intuitive to use user action. This triggers an interaction between the Multi-Agent System and the visual management application, resulting in a migration of a mobile agent between different Agent Nodes. The example shows how the software agent metaphor, used in AOSE concepts at design time, can be sustained and elaborated in runtime tools for Multi-Agent Systems.

1. INTRODUCTION

Distributed Multi-Agent Systems (MAS) are, in general, rather complex systems used to solve complex tasks. Physical distribution as well as large numbers of agents become serious issues when monitoring running MAS. Classical monitoring tools, applying user interface elements like log file outputs and tables of entities, are unintuitive and become confusing with rising numbers of elements. The software agent approach provides concepts for dealing with distributed and highly dynamic infrastructures, in which entities are added and removed in large numbers and at different locations. An user interface (UI) has to keep up with these aspects, giving the user overview and manipulation abilities to assert control over the MAS at runtime. Text based interaction, for displaying system information or issuing commands, is less than optimal. Simple user interactions can provide easy access to complex functionalities and the dynamics of MAS allow agents to react to infrastructure changes as conducted by an administrator.

We propose to provide insight into a running MAS infrastructure, by providing an approach of intuitive interaction for MAS administrators relying on graphics, common metaphors and real-world knowledge of users. Major elements of this approach are user interactions that allow an user to change his perspective or manipulate entities in the running MAS, e.g. stopping a specific agent. In this paper we focus on agent migration by Drag-and-Drop as an example of a concept, design and implementation of such an user interaction. Agent migration is well known concept that can be used to achieve goals like load balancing or administrative tasks. This example provides an excellent opportunity to show the connections between a simple metaphor known to any user, the concept of how this metaphor is integrated into the UI and the implementation details of the resulting software components. The following user interaction approach example relies on the Advanced Structured Graphical Agent Realm Display (ASGARD) [11] which provides the foundation for the MAS visualization. The example implementation uses the JIAC V framework and is implemented in Java. In JIAC V agents can be mobile to support several types of migration (weak/strong) and can be cloned in order to create similar redundant agents. In ASGARD agents are represented as boardgame play figures that can be dragged and dropped from node to node, similar to a real life boardgame.

The details of this concept and implementation are described in the following sections and are structured as follows: The next section provides an overview of related work in MAS monitoring. This allows a more accurate placement of our approach in comparison with given concepts. Related work is followed by a general concept description providing details over the different aspects of this approach. Subsequently the prototype implementation describes how the concept was realized and finally a conclusion section discusses evaluation, results and future work.

2. RELATED WORK

2.1 Shell-based Monitoring and Interaction

The classical and most frequently used approach to interact with running applications by developers is the use of command shells and their input and output functionality. These command shell applications are usually provided by the operating system. Their main advantage is that command shell output (and by a lesser margin input as well) is easily implemented in any application. This is contrasted by the fact that command shell interaction is not intuitive. There is no way for the user to judge the importance of a textual log output on first look, and keyboard commands (if available) tend to be of a rather cryptic nature. Furthermore, as waiting for keyboard input might block an application or is rather complex to implement in a non-blocking way, most developers only offer the process of stopping an application, reconfiguring it and restarting as a mean of interaction.

Those disadvantages of shell-based interaction get even more significant in a distributed MAS, as the usual approaches either require the developer to watch a dedicated

shell console for each processing entity in the MAS or try to differentiate the log outputs of each entity in a single console. Providing live interaction for the MAS is almost impossible by those means.

2.2 Tools for Post Mortem Analysis

Several monitoring tools that use a more visual approach are available for various MAS implementations. One of these is the **ADAM3D** [4] tool that visualizes interaction between agents on the base of 3D technology, using the third dimension as a temporal axis. The **Brahms Agent Framework** [9] provides it's own visualization tool as well, which visualizes property changes and communication between agents over time. Both of these tools rely on log output that is collected in a database, so they can only be used for post mortem analysis of a MAS and thus cannot provide any sufficient live management functionality.

2.3 Live Management Tools

There are few generic solutions for the problem of providing live management for a MAS so far. The **JIAC Node Monitor** [6, p. 126] for the JIAC Agent Framework [3] is a first attempt at solving this problem and can be seen as the predecessor to the ASGARD concept, but is limited to a single *Agent Node*[1] and thus does not represent the distributed nature of a MAS. It's main disadvantage is the limited scalability of it's visualizations, and it's management functions were not intuitively designed. However the Node Monitor shows that a visual monitoring and management application does improve the workflow during the implementation process of a MAS application. A developer can use the existing monitoring solution to check his entities instead of having to implement his own interface for monitoring and management.

3. CONCEPT

Herein metaphors for the user interaction, the migration concept and the management aspects for our approach are described.

3.1 Metaphoric understandability

A key requirement in the creation of an easy-to-use management solution is that it provides an intuitive interface, so that the user will understand visualizations and interactions because of prior experiences and knowledge. To achieve this effect, a common attempt is the use of metaphors [1]. Metaphors provide a mental bridge between the abstract nature of the software "world" and concrete objects in the real world. As this link is essential for understandability, a good metaphor should always have a lot in common with the represented software entity.

In the JIAC V[3] framework, there is a set of basic entities that provide the infrastructure (see section 4). To create a visualization of these structures in ASGARD, the most important JIAC V entities are replaced with metaphors according to their role. *Agents*, as the entity that provides all application-specific processes in the MAS, can easily be visualized by an abstract human figure such as a play figure from common board games. *Agent Nodes* as the providers of a runtime environment for agents are consequently represented by rectangular platforms, as a "floorboard" for agents. Agents are placed on these platforms to make the hierarchic relationship between agents and Agent Nodes instantly visible. *Communication* is a straightforward choice as well. The image of a letter envelope is one of the most common metaphors to indicate a message. To add even more understandability and the possibility to trace the communicating entities, a path between those entities is shown.

States of entities are a bit more diverse. Some states like the life cycle of an entity can best be visualized by coloring parts of the corresponding metaphor, such as an agent head being colored green or red to indicate a running or stopped state. This choice of colors is straightforward as it relates to the common experience with traffic lights. Other states and capabilities are better visualized by adding icons to an entity, such as the suitcase icon in figure 1 indicating migrateable agents and nodes that support migration.[2]

3.2 Migration

The migration of agents is a process which allows an agent to move from a source node to a target node by sending a mobile agent description as message. This process extends the agent's life cycle. Thus a new class of agents are established on JIAC V platforms – mobile agents.

Mobile Agents extend common agents on JIAC V. A mobile agent is aware of its own configuration, library dependencies and feature requirements.

A mobile agent has the capability to invoke his own migration to an other Agent Node, but is not designed to avoid his migration. A mobile agent creates an abstract description of itself, so called *Agent Image*.

This description is a message that can be transmitted. The agent image differs from a simple *Agent Description* which contains agent name, unique identifier and agent address for communication.

In JIAC V we distinguish between different modes of migration. In the first step the process always creates an agent image. The migration modes vary from each other in how thorough the agent image reflects the migrating agent. The agent image has to contain enough data to perform the specified mode of migration, e.g. for starting with a clean state, or one that maintains the complete state the agent was in right before the migration was triggered.

In the process of migration the *Agent Node*'s task is well-defined: It has to handle the migration on the agent node level. The agent nodes supporting mobility are grouped by using a reserved message bus. This bus is supposed to find other nodes with mobility support and exchange their addresses. It is furthermore an abstract facility. Intermediate agent nodes on the route of a message (e.g. nodes acting as a network gateway) are hidden.

While migrating an agent, the two involved agent nodes communicate directly to move the agent from one to the other. The individual addresses of the partner node is known from the message bus.

The migration process (see figure 5) starts with the send-request of the mobile agent to the source Agent Node. The request consists of the complete requirements of the mobile agent. Alternatively an upper authority can request the migration for a distinct agent, e.g. a node management component.

The target node is able to accept or reject this request. In

[1]see section 3.1 or [3]

[2]A more detailed discussion of JIAC entities and their metaphors in ASGARD can be found in [11].

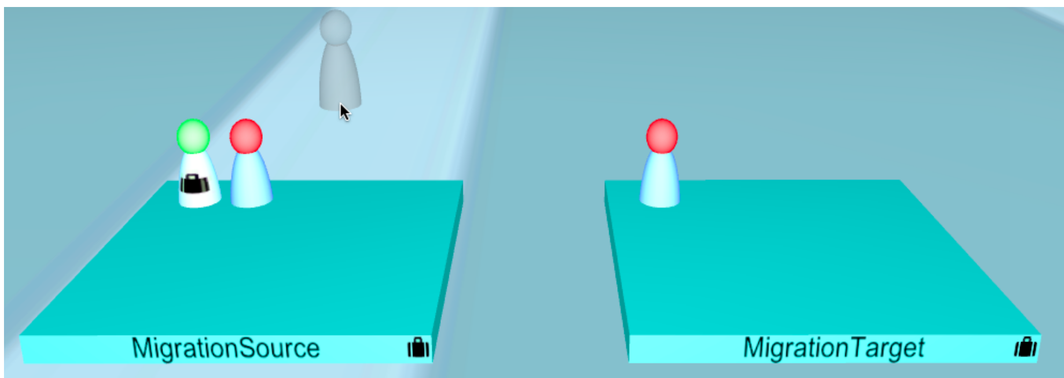

Figure 1: The user drags the migratable agent

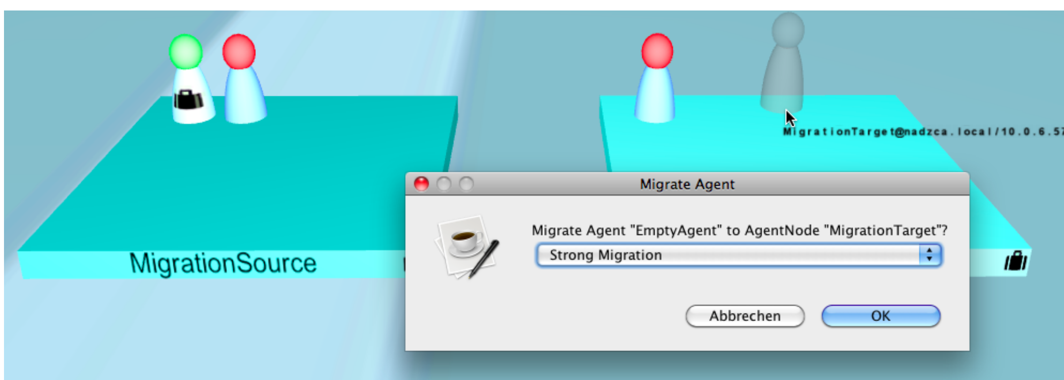

Figure 2: The user drops the migratable agent and triggers the migration

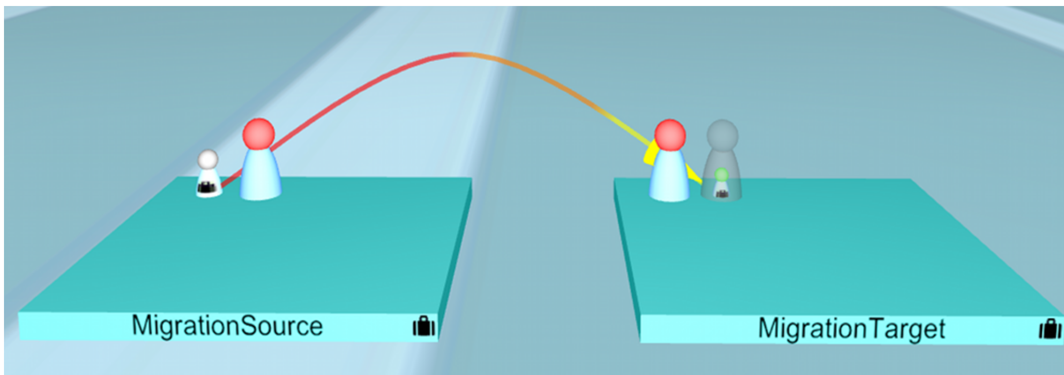

Figure 3: The migration process is animated

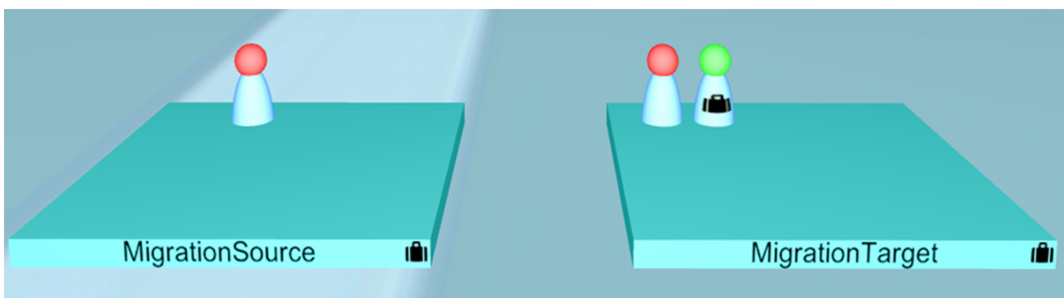

Figure 4: The migrated agent now runs on the target node.

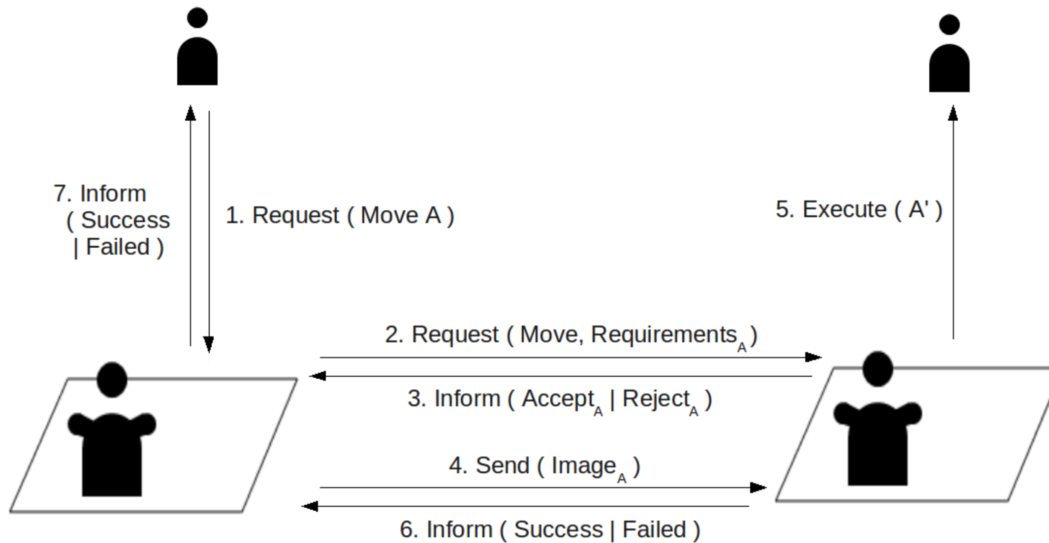

Figure 5: Interactions between Agent Nodes and agents for migration process

case of an acceptance the agent can be transported to the target node. In doing this, the complete agent image will be send from source to target node. After completing transmission the agent will be reconstructed from the transmitted image. The source node will be informed about success or failure of the transportation process. Finally the agent receives feedback about the whole migration process. In case of migration without cloning the original agent is shut down, which completes the migration process, whereas in case of cloning, the process is already completed upon receiving the success or failure message. During the whole process, notifications are send over the management interface that contain information about the current state of the migration process. This enables a monitoring tool like ASGARD to keep track of the migration process.

The visualization of this migration process makes use of the metaphors mentioned in section 3.1. The existing metaphors for agents and Agent Nodes are used as a base to visualize this process. To show the initiation of a migration process from one node to another, a translucent copy of an agent shape, a "ghost agent", is placed on the target node first. The migrating agent's representation on the source node is then animated to shrink, whereas the ghost agent is filled by a regular agent object growing inside of it, as visible in figure 3. As this visualization alone does not emphasize the communication aspect of the migration process, a path between the positions of the agents on source and target Agent Node is shown as well. This path indicates the direction of the migration and makes it easy to spot the corresponding partners in situations with a lot of migrations occurring.

3.3 Management using Drag-and-Drop

Distributed MAS are usually long lived infrastructures providing services for users or other software systems. Demonstration, maintenance or other reasons sometimes demand the ability to migrate agents from one node to another in a running platform. However, short of writing a quick program to trigger such a migration, there generally are no direct manipulation options for a MAS administrator that could accomplish such a trivial task (trivial once the rather complex process of migrating a mobile agent is already implemented). Thus a simple user action is required that allows a user to trigger migrations at will and the well known Drag-and-Drop metaphor is an obvious choice.

The Drag-and-Drop process has evolved to a standard feature in all modern user interfaces. It is based on the metaphor of moving an object from a source location to a target location, and thus used for all kinds of processes that change the physical or logical location of an entity, such as moving files between folders. The usual implementation is that the user uses the mouse to select an object and move it to the target while holding the mouse button down (Drag) and releasing the button once the target is reached (Drop). The common acceptance of Drag-and-Drop in user interfaces makes it an optimal metaphor to manage migration processes as well, as a migration process in a MAS is characterized by being a location change of an agent.

To make use of this, ASGARD offers control over migration by dragging an agent from the source node onto a target node using the mouse. Migrateable agents are easy to identify for the user by the suitcase icon on agents, showing the migration ability, Agent Nodes with support for migration are similarly marked. To indicate that an agent is being dragged, a ghost agent is attached to the mouse cursor and moved along with it (see figure 1). Once the agent has been dropped onto a target Agent Node that supports migration, a standard GUI dialog (figure 2) asks the user which kind of process (Strong/Weak Migration or Cloning) is required. ASGARD then initializes the migration process and visualizes it's progress in the same way as an internally triggered migration would be visualized (figure 3).

The separation of the interaction and visualization has the advantage of offering a base for automatically visualizing errors. If an error occurs during the initiated migration process, the migration animation will not be shown and the user will see that the migration process was not executed as anticipated. This concept can even be extended to offer

detailed information about the occurred error, by reading out the involved entities' properties and presenting them in a textual or graphical way to the ASGARD user.

3.4 Other interaction metaphors

In it's current state, ASGARD offers several other concepts of interactions using metaphors. One of them is the idea of showing a greater level of detail when using the zoom feature on a selected entity. This matches the users knowledge. He can see far more details of an object if he watches it from a close point of view than from far away. The adaptive level of detail has the advantage of solving a scaling problem as well, as a visualization of every detail in every entity would be way to complex and space-consuming to still be intuitive and understandable.

Other planned features is the use of Drag-and-Drop between ASGARD and a visual editor for agent and Agent Node configurations (AWE [7]) to deploy new agents in the MAS, and a similar way to remove agents and Agent Nodes from the running system. The latter could be done by dragging an entity onto a waste basket, as this metaphor for removing an object is common in current operating systems as well.

4. IMPLEMENTATION

In order to implement the concept of Drag-and-Drop migration the Java based Intelligent Agent Framework version V (JIAC V) was used. JIAC V combines agent technology with a service-oriented approach and provides a wide range of basic functionalities for agent deployment, communication, management and dynamically changing distributed environments. JIAC V MAS consist of a hierarchical structure as shown in figure 6.

A Platform consists of a range of different Agent Nodes that provide the environment for the execution of agents. An Agent Node is roughly equivalent to a Java VM with an agent infrastructure. Individual Agents are implemented as Java objects in an Agent Node and consist of several core components and optional Agent Beans. These beans are used to provide concrete functionalities as plug-in components for agents. The JIAC V foundation is used in a wide range of projects, ranging from service delivery platforms to simulation environments. The following sections elaborate on three specific components of the JIAC V framework relevant for Drag-and-Drop migration.

4.1 Migration Implementation

JIAC supports basic migration capabilities by adding mobility support to the Agent Nodes and instantiating a new mobile agent. JIACs Agent Migration API provides strong migration. The MAS developer only has to use the Java annotation `@Migratable` for migratable fields. Only the stateful information in the Agent Beans has to be marked. There are three ways for the developer to mark the migratable agent properties in an Agent Bean, in order to provide a solution that is usable in all combinations of field access levels. The Developer does not have to stick to a fixed naming scheme for getter and setter methods.

In order to realize strong migration an agents state has to be archived. We do not transfer the whole execution stack of an agent out of a Java VM. The JIAC V API collects all stateful information of all agent properties. The agent engineer marks all stateful information about the agent beans belonging to the mobile agent.

As described in section 3.2 an agent creates its own image. The agent is aware of its configuration and corresponding implementation, e.g. Java classes. Creating a complete image of an agent for strong migration is a complex and expensive process. Due to this we adopted Java Annotations as a simple way for realizing migration support. Each Java field, constituting a migratable state, is marked with the annotation, which is only being evaluated in the case of strong migration. For this purpose, every Java class implementing an agent feature requires a scanning for annotations. This process will be done if an agent image is needed within a strong migration.

A mobile agent is equipped with a scanner, validator and collector for agent properties. For every Java class related to an agent feature the scanner collects Java fields annotated with `@Migratable`. The output of the scanner is a list of Java fields comprising field names and field types. The scanner furthermore determines potential setter and getter methods. Subsequently, each agent property requires validation.

The validator uses the list of collected Java fields. Every field will be approved to be accessible and serializable. A field is accessible, if it is public writeable or it has public *set* and *get* methods. Java fields which do not pass the validation will be dropped, while we call the remaining ones *Agent properties*. These are accessible and serializable properties which can be read and set for the agent.

Finally a collector iterates over the validated agent properties and collects their current values. The collector creates a list of agent properties, comprising their names and values. The set of all collected agent properties constitutes the reconstructible state of the agent.

4.2 Management Interface JMX

The binding element between the migration implementation in the JIAC V Framework described above and the ASGARD visualization implementation described below is the JIAC V management interface. Since JIAC V is implemented in Java, the application of the standard management interface concept JMX [10] was an obvious step. This adds the ability to access the JIAC V infrastructure and analyze an agent Nodes internal data. It can be used by standard tools like JConsole, as well as by proprietary implementations like the ASGARD tool. Standard infrastructure elements in JIAC V, like Agent Nodes, agents and their Beans, provide JMX based access methods that allow remote access to their internal state, data and some functionalities. Agent Nodes are located in the LAN subnet by multicast messages that use a special management channel on the message bus. Last but not least remote JMX clients can register for specific notifications in order to prevent inefficient polling mechanisms.

JMX relies on managed beans (MBeans) that effectively are Java interfaces. By implementing such an interface a class gains the ability to register with a local management component (also called "agent" in JMX terminology) and provides remote access to the attributes and methods (termed "services" in JMX) and notifications through the JMX "agent" component. JMX based client classes can then access the specified interface remotely, usually through the RMI protocol. The JIAC V framework provides a range of standard JMX clients for specific infrastructure elements like Agent Nodes, agents, etc., that ASGARD utilizes as described in

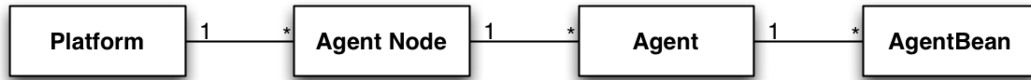

Figure 6: Basic structure of entities in the JIAC V framework.

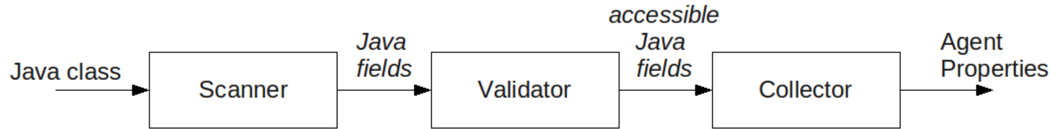

Figure 7: steps to scan Java classes for migratable agent properties

the following section.

4.3 ASGARD

The ASGARD application provides the user interface to the management functions of a JIAC MAS. Using the JMX-based management interface to retrieve information about the current state of the MAS, ASGARD generates a metaphoric visualization at runtime, which is used to provide information to the user as well as a base for user interaction with the system. To create the visualization, ASGARD has to locate the JIAC entities in the network infrastructure. This feature is currently implemented by receiving multicast discovery packets and reading out Java RMI registries. The information received this way consists of JMX service URLs, which are used to connect the management clients to the MAS entities.

As two-dimensional graphic engines and raster images have the problem of not being scalable quite well, ASGARD uses 3D graphics to visualize the structure of JIAC V applications. For this purpose, a 3D engine called *Java Monkey Engine (JME)*[5] is used. It offers bindings to the OpenGL API for most common operating systems, so that ASGARD can be used on a large variety of systems. Furthermore, JME provides a lot of functionality for creating and managing three-dimensional objects and for user interaction. These features turn JME into a solid base for a graphical and interactive application.

ASGARD uses the data from polling the management clients and receiving JMX notifications to generate a tree structure of object instances containing the metaphoric representation of a connected JIAC MAS. The reason why both polling and notifications are used is that the JIAC V management interface provides notifications only for events and property changes that are quite frequent to happen. This provides instant knowledge of their occurrence. Properties that are changed rarely or never can be retrieved by polling with a low frequency. The combination of those techniques offers a good tradeoff between network load and up-to-date information about the managed system's state.

The visual representation objects are built up in an Model-View-Controller (MVC) [8] pattern style to provide easy adaption and a separation of data, visual implementation and interactions. This separation is performed by creating three different objects for each entity. The *entity controller* manages creation and deletion handling for the object and polls the management interface for data. Furthermore, it registers as receiver for both JMX notifications and user interaction events such as mouse clicks. The data retrieved over the management interface is stored in the *data model*. It acts as a buffer for value properties of an entity and provides the base to generate a visualization as well as for a textual presentation of the data in a property table. The *visualization* is generated and managed in a third object that represents the "view" part of the MVC pattern. This object uses the data from the data model to generate and adapt the visualization to the current state of the entity.

As ASGARD uses a three-dimensional space for the visualization, finding the entity that the user is currently pointing at with the mouse cursor has to be done by an algorithm called *ray intersection*. This is done by sending a virtual ray from the camera position in the direction of the mouse cursor into the scene and checking which object's bounding box[3] is intersected by it. ASGARD then forwards mouse over and mouse click events to the foremost entity that got intersected, so that mouse events get handled by the corresponding entity controller class.

The visualization of a migration process is based on the JMX notification mechanism. The Agent Node controller instances in ASGARD register as notification receivers. The migration implementation sends notifications containing the current state of the migration process and identifiers for the involved Agent Nodes and agents, Upon receiving those migration notifications, the representations for the involved entities are identified in ASGARD's internal entity tree. The visualization handler instances then create the visuals and animations which are shown in figure 3.

By extending the user action event handler mechanism to special Drag-and-Drop events, ASGARD is able to provide the functionality to offer migration per Drag-and-Drop to the user. The drag animation is implemented pretty straightforward by moving a "ghost" copy of the entity according to the mouse cursor motion (see figure 1). If the user releases the mouse button, the entity that was below the dragged entity gets identified by ray intersection and receives a drop event that contains a reference to the dragged object. Thus, the drop handler can provide different handling for different entities. In case of the dragged entity being a migrateable agent and the agent Node supporting

[3] The ray/box intersection and the use of bounding volumes for collision checks are common algorithms in 3D computer graphics. Intersection checks are usually performed by solving linear equations to find the intersection point or plane. Further details can be found in [2].

migration[4], the Agent Node controller will trigger the migration process (see figure 3) by calling the appropriate method of the JMX management system. The migration process is then executed in the same way as if an internal event in the MAS had triggered it, so that the same visualization algorithm that shows all occurring migration processes in the MAS will provide a visualization without further adaptions.

5. EVALUATION AND CONCLUSION

5.1 Evaluation

The implementation of a visualization for JIAC V's migration feature and the subsequent adding of the Drag-and-Drop migration management has shown to be successful during an initial institution-wide testing process. The test users were able to instantly connect the visualized process with migration. The migration visualization shows clearly when a migration process is occurring in a MAS application, the current state the migration process is in, and which entities are involved. Using ASGARD during debugging processes has helped to identify migration-related problems such as failed migrations or agents migrating onto a wrong target node.

The implementation of triggering a migration using Drag-and-Drop has proved successful as well. Instead of having to add functionality to offer a user-triggered migration manually to each JIAC V application, developers can now rely on being able to test migration processes by just dragging agents between nodes in ASGARD's visualization of their implementation. As this feature is available in the JIAC V environment for all migratable agents as part of the management interface without any further work for the user, it saves a lot of implementation effort for testing and demonstration processes.

A standardized test of the whole ASGARD application with a larger and more diverse group of users is still to be conducted once the prototype implementation reaches a stable state.

5.2 Conclusion

In this paper we presented an example for mapping user actions to agent infrastructures, the Drag-and-Drop Migration. We motivated why graphical user interfaces provide added value for administrating MAS and which major technologies are involved in our example. We compared our approach with related work and gave a deeper insight into the concepts of the metaphors, migration and Drag-and-Drop concepts used herein. Afterwards we provided more insight into the implementation details of our example and described the different components involved. We conclude with a results evaluation and provide some hints for future work in this concluding section.
The resulting concept shows how user actions can be used to trigger complex operations in a MAS infrastructure, through the application of graphical user interfaces. This requires carefully selected metaphors and an extensive framework to rely on. The result is a visual interaction method that is convenient for administrators as well as intuitive to understand for other observers, e.g. for demonstration purposes.

[4]Both of these abilities are identified using JMX when the entity data and information is gathered.

5.2.1 Future Work

As the concept and initial implementation of Drag-and-Drop migration in ASGARD has proved to be successful during everyday use, more management functionality should be offered to users in a similar way. The integration of agent deployment and removal by Drag-and-Drop already mentioned in section 3.4 are two of them. Other interaction metaphors will be used for life cycle and property influence of entities and to make visual navigation and locating certain entities easier to handle.

6. REFERENCES

[1] S. Diehl. *Software Visualization - Visualizing the Structure, Behaviour and Evolution of Software.* Springer, 2007. ISBN 978-3-540-46504-1.

[2] M. Gomez. Simple Intersection Tests For Games. Gamasutra online article, www.gamasutra.com, October 1999.

[3] B. Hirsch, T. Konnerth, and A. Heßler. Merging Agents and Services — the JIAC Agent Platform. In R. H. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 159–185. Springer US, 2009.

[4] S. Ilarri, J. L. Serrano, E. Mena, and R. Trillo. 3D Monitoring of Distributed Multiagent Systems. In *WEBIST 2007 - International Conference on Web Information Systems and Technologies*, pages 439–442, 2007.

[5] JME Development Team, http://www.jmonkeyengine.com. *Java Monkey Engine*, 2009.

[6] J. Keiser. *MIAS: Management Infrastruktur für agentenbasierte Systeme.* PhD thesis, Technische Universität Berlin, September 2008.

[7] M. Lützenberger, T. Küster, A. Heßler, and B. Hirsch. Unifying JIAC agent development with AWE. In *Proceedings of the Seventh German Conference on Multiagent System Technologies, Hamburg, Germany.* Springer, 2009.

[8] T. Reenskaug. Models-Views-Controllers. Technical report, Xerox-Parc, 12 1979.

[9] M. Sierhuis, W. J. Clancey, and R. van Hoof. Brahms - a multiagent modeling environment for simulating social phenomena. In *First conference of the European Social Simulation Association, Groningen*, 2003.

[10] Sun Microsystems, Inc. *Java Management Extensions (JMX) Specification, version 1.4*, 11 2006.

[11] J. Tonn and S. Kaiser. ASGARD - A Graphical Monitoring Tool for Distributed Agent Infrastructures. In *8th International Conference on Practical Applications of Agents and Multi-Agent Systems, University of Salamanca, Spain*, 2010.