HAI – A Human Agent Interface for JIAC

Sebastian Ahrndt, Marco Lützenberger, Axel Heßler, and Sahin Albayrak

DAI-Labor, Technische Universität Berlin, Germany
Ernst-Reuter-Platz 7
10587 Berlin, Germany
{sebastian.ahrndt|marco.luetzenberger|axel.hessler|
sahin.albayrak}@dai-labor.de

Abstract. There are many different application frameworks, which accelerate user interface development by simplifying repetitive and time consuming tasks. Most of these frameworks follow the widely accepted Model-View-Controller (MVC) architecture. Although, the existing frameworks are optimized for the implementation of object-oriented applications. The special features and possibilities offered by agent applications are not supported. Within this work, an architecture is designed, which allows the integration of any user interface in agent applications. As an extension of the MVC architecture, the advantages of agent-orientated software engineering will be combined with the advantages of the existing application frameworks. This structure provides the base for the Human Agent Interface, which allows the integration of any user interface in JIAC V agent systems.

Keywords: Human Agent Interaction, Interface, Agent to non-agent interoperability

1 Introduction

The implementation of high-quality user interfaces (UI) is considered to be an essential phase of almost any software project. The importance of UIs and the complexity in developing them has been recognised long since. In an early work, *Myers* and *Rosson* [10] have analysed the implementation of user interfaces in detail and estimate the expenses on up to 50% of the total budget. In order to manage and accelerate the UI development, a large number of approaches and technologies have been presented so far. Support is mainly provided by so-called *Application Frameworks*, which simplify repetitive and time consuming tasks by providing tools, structures and reusable artefacts. Yet, as comprehensive and sophisticated these frameworks are, their principle is usually geared towards object-oriented structures and neglects support for other paradigms.
Due to their distributed nature, agent-based systems have an increased demand for user interfaces. Each involved agent usually requires an interface for configuration, management and observation purposes and, as the benefits of application frameworks are indisputable, we invented a way to extend common

agent-oriented software development with application framework concepts and exploit the advantages of both techniques.
In this paper, we present this approach. We start by describing our concept (see Section 2), which we evaluate afterwards (see Section 3). Subsequently, we compare our work to existing solutions (see Section 4) and finally, we wrap up with a conclusion (see Section 5). This paper is based on the diploma thesis of the first author [2].

2 Approach

The objective of our work was to facilitate interaction between user interfaces and multi-agent systems (MAS) by integrating application frameworks into the agent-oriented application development. When analysing related works, we were able to identify two main directions. On the one hand, agents were used in combination with existing frameworks. To make this approach work, the agents were geared towards the respective framework. On the other hand, we identified solutions with particular support for agent-oriented software development. These solutions were usually restricted by the applied technology. As an example, consider web-based approaches. Web orientation enables a standardised and easy distribution of UIs and supports a large number of different (mobile) devices, yet, the implementation is tied to web based programming languages as well as to the unflexible request-response model. The main intention of our work, was to counter such limitations.
We implemented our concept for the latest version of *JIAC V*, which we introduce in the following. Even though we made use of a particular agent framework for our implementation, our concept is not limited to it, but can be applied to many other agent platforms.

2.1 The Java Intelligent Agents Componentware (JIAC)

JIAC is a Java based agent framework which has been developed within industry- and government-funded projects at the *DAI-Labor* since 1998 [3]. In its latest incarnation, *JIAC V* [6] combines agent technology, scalability and performance and allows to execute hundreds of agents on a desktop computer. The modular architecture includes conformity to standards, extensibility, security mechanisms and combines service-oriented concepts with agent-oriented approaches. Agents can be implemented using Java or the *JIAC Agent Description Language* (*JADL++*), *JIAC*'s scripting language [6]. In *JIAC*, an agent's behaviour, its capabilities and its functions are encapsulated by so called "AgentBeans". The entire development process is supported by a comprehensive set of tools.

2.2 HAI in a Nutshell

The Human Agent Interface (HAI) has been developed with the objective to facilitate interaction between user interfaces and multi-agent systems. We derived

the name “Human Agent Interface” directly from the purpose of our development. In order to remain independent from a specific UI technology or a particular agent framework, we arranged HAI to act as some kind of middle layer between the UI and the MAS. The advantage of this architecture lies with the clear separation between agent-specific parts and those from the user interface, such as JSPs, Silverlight pages or XUL, Flex components, to name but a few. Both, details and specific properties of the agent layer are hidden from the user interface. For the UI developer, the application appears as a monolithic system, whose functions are accessible via an API.

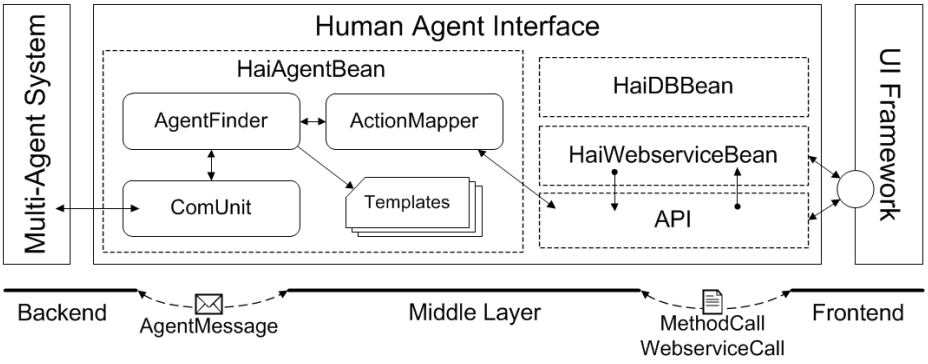

Fig. 1. The HAI as middle layer mediating between UI and MAS.

Figure 1 emphasises the role as a middle layer and illustrates the HAI with some of its components. It also shows one of the main tasks: Converting UI request into agent messages and vice versa. In compliance with JIAC’s principle of modularity, each capability is implemented as an AgentBean. Consequently, we implemented our HAI concept as JIAC V agent: The *HAI-Agent*. The loose coupling between capabilities and agent allows for custom and streamlined context adaptations, since unused features can be removed easily.

2.3 Integrating agents into MVC

Using application frameworks to implement user interfaces offers many advantages for the development process. Nevertheless, as an objective of our work, we wanted to ensure, agent and UI developers do not influence each other. For this purpose, we made use of the MVC paradigm, since it logically separates an application on programming level.
In an MVC application the logic is implemented within the controller. When it comes to agent applications, the agents are usually dedicated to some application goal. The Human Agent Interface combines both principles and allows for the integration of agents into applications that comply with MVC. Since the application logic is hosted within the agents, a UI developer is not affected, when

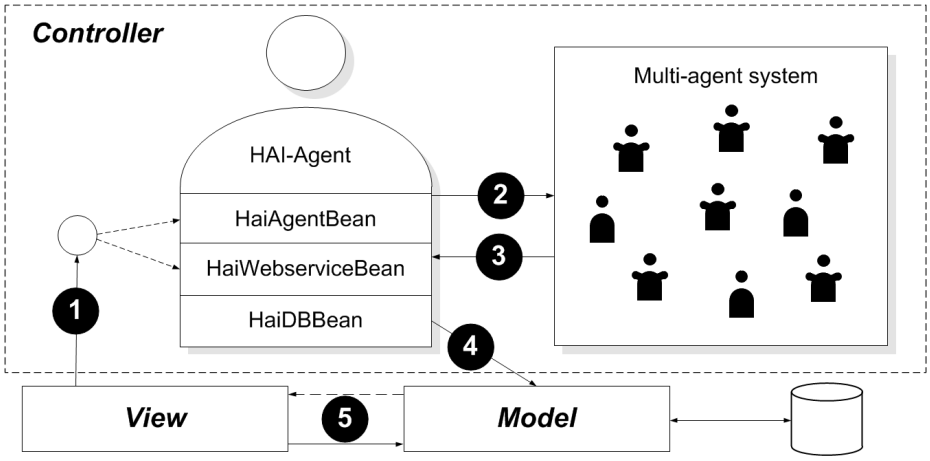

Fig. 2. MVC architecture with extended agent-based controller.

the controlling components are changed. Figure 2 shows the MVC architecture including Human Agent Interface and the MAS. The controller is modified in a way, that the HAI-Agent can act as mediator and the application logic is realised by the MAS. For this, incoming requests (1) are forwarded to the agent-system. Requests are either triggered by method calls and directly sent to the HaiAgentBean or triggered by web service calls and sent to the HaiWebserviceBean which forwards them to the HaiAgentBean. If necessary the HaiDBBean is involved to provide required data from a connected database. The request will then be converted into an agent message, with additional data. The HAI-Agent proceeds with the search for agents which are able to process the request. A definition of the nature of a request is given by the user interface as parameter to the HaiAgentBean. If a fitting agent is located, the previously generated agent message will be forwarded (2). Otherwise a failure code will be returned. The advantage of this principle is the transparent communication between UI and the MAS for the agent developers. Communication from the user interface can be handled just as any other communication. The focus in implementing agent systems thus remains on the agents themselves and does not additionally comprise user interface topics. After the agent has provided an answer to the redirected request, the result is sent back to the HAI-Agent (3). The HAI-Agent then updates the data model if necessary (4) and notifies the view on potential changes, which are applied automatically (5).

For web interfaces the structure is slightly different since the underlying request-response model does not support active views. The controller component has to specify a suitable view for the modified data — this structure is commonly called: Model 2 paradigm [5]. If HAI is used in this context, incoming data is stored in combination with an ID, which can be retrieved by a third controller component (such as a servlet) which defines the associated view.

3 From theory to practice

We evaluated our work, within a government-funded project, in which we developed a complex graphical user interface and designed HAI to act as an interface between the UI layer and the MAS. The objective of the UI was to control and to visualise the results of an agent-based traffic simulation framework, which adds perceptions and attitudes of drivers to the simulation process [9]. To emphasise the operation principle of HAI, we describe some of the steps, done.

```
def initMap = {
  def i = new Invocation(name: "initMap", invocationId: null)
  i.invocationId = hai.invoke(MapServiceAgent.INIT, BERLIN)
  if(!i.save()){println "cannot save invocation id"}
}
```

Listing 1.1. Causing the HAI-Agent to search a spezific action.

For our work, we used *GRAILS* [12] as application framework. Since GRAILS is based on *Groovy* we had almost no trouble in using HAI for this project (`hai`). For the simulation framework, a simulation environment (city) had to be selected, first. Since different maps had different complexity and size, we were not able to define a suitable initialisation time-out. Instead, we applied asynchronous communication and realised the initialisation process by active waiting. Listing 1.1 illustrates our implementation, which causes the HAI-Agent to search for an agent which offers the `MapServiceAgent.INIT` action. The HAI-Agent converts the method call into an agent message, which is forwarded to the agent-system. After forwarding the message, an ID is generated, stored and returned to the UI as confirmation and in order to allow the UI actively check for answers. Since the simulation environment is vital for the simulation itself, the UI waits as long as an answer arrives or a time-out expired. Listing 1.2 shows the method which is called periodical to check for an answer.

```
def lookForAnswer = {
  def inv = Invocation.findByName("initMap")
  if(inv != null){
    if(timeout()){
      hai.deleteInvocation(inv.invocationId)
    }else{
      def res = hai.lookForInvocationResult(inv.invocationId)
      if(res != null){
        inv.delete(flush:true)
        render(view: "/settings/index", params:["map":res])
      }
    }
  }
}
```

Listing 1.2. Periodical check for an answer.

In order to support the many configuration parameters of the simulation framework, we made use of so called "configuration objects", which we stored within a database. Listing 1.3 shows a method (*findObject*) which uses the HaiDBBean to send data from the user interface to the agent-system. Since both, the agent-system and the user interface use the same data model, no translation was implemented.

```
def sendMapWeight = {
  hai.invoke(MapServiceAgent.SET_WEIGHT, hai.findObject("MapWeight", params.id))
  flash.message = "Preferences saved"
  render(view: "/settings/index")
}
```

Listing 1.3. Sending data from UI to MAS using the HaiDBBean

4 Related Work

To distinguish our work from others, we performed a comprehensive survey on existing approaches. We identified several approaches, but only two of them were generic and can be classified as frameworks: The *JACK WebBot* [1] and the *Jadex Webbridge Framework* [11]. Both of them support the development of web-based agent applications. The Jadex Webbridge Framework facilitates application development in compliance with the Model 2 paradigm. It allows the agent developer to do his work without knowledge on web development and by extending a provided controller component. The JACK WebBot is an extension of the JACK [7] framework and allows for the development of web applications by using JACK agents. JACK WebBot extends the Java Servlet API and thus requires a servlet container to provide user interfaces. To interact with the user interface JACK agents have to implement special interfaces.
Furthermore, we were able to identify several ad-hoc solutions. The *JADE* Framework [4], for instance, is equipped with the *JadeGateway* [8] and the *JADE GUI Agent* [15]. Both are concepts to furnish agent applications with servlet based web or SWING UIs. JIAC is equipped with the *Alter-Ego* [13] concept and the *MAMS* (Multi Access, Modular Services) [14] principle. Both concepts facilitate communication between UI and MAS, yet, most of the technical details have to be implemented by the agent developers.

4.1 Discussion

In our survey, we identified only two approaches capable of furnishing agent systems with user interfaces: The Jadex Webbridge Framework and the Jack WebBot. Both approaches facilitate the development of web-based agent applications. While the JACK WebBot has been tailored for JACK agent applications, the concept of the Jadex Webbridge Framework is a generic one and can also be realised with other agent frameworks. As a matter of fact, both solutions restrict the agents in their proactive behavior. Without an open request, an agent cannot start any interaction with the user. This function is provided by the Human Interface Agent. At the same HAI is not specialised to one type of user interface technology.
Nevertheless, there are other features that are not supported by HAI. The Webbridge Framework for instance allows to create agents for a request and also to withdraw them when they are no longer needed. This is an important feature,

especially for resource management. The JACK WebBot is able to read and manipulate requests. This feature is used to increase the security of web sessions. The modular concept of HAI helps to solve some of these shortcomings in the future. This is one reason why we implemented HAI as JIAC V agent.

5 Conclusion

In this paper, we introduced a concept which enables and simplifies the development process of agent applications with user interfaces. Based on a survey similar approaches, we can say that there are a lot of ad-hoc but only a few generic solutions. These solutions were usually restricted by the applied technology. To counter such limitations and to bridge the gap between user interfaces and agent systems we introduced an extra layer: The Human Agent Interface (HAI). As a mediator, it performs all operations which are necessary to hide the details and specific properties of the agent layer from the user interface layer. To achieve the integration of this extra layer we introduced a concept which is based on MVC and integrates agents in applications. This approach allows developers to continue using application frameworks, which provide tools and functions that simplify repetitive and time consuming tasks to speed up the implementation of user interfaces. The agent-based MVC approach refines the controller component by splitting it up into the HAI and the agent system (for web interfaces there are three parts required: The servlet, which forwards request to HAI, the HAI itself and the agent system). The HAI is responsible for the conversion of user-requests into agent messages, inserting the required data and sending the message to a suitable agent in the multi-agent system. The actual program logic is performed by the agent system. User interface and MAS are kept separate. For this reason they can be implemented (mostly) independently from each other.
Further, a short insight into UI development with the Human Agent Interface has been given. The implemented HAI not only consists of the components described in the concept. We rather implemented a framework, which allows for monitoring, configuration and extensibility of the Human Agent Interface.

6 Future Work

A lot of research has been done in the domain of interface agents, to improve the interaction between humans and computers. But there are only a few approaches to equip agent applications with user interfaces. The Human Agent Interface is such a solution. However, it relates to the use of application frameworks, which are designed for object-oriented applications. To combine the advantages of agent-orientated software engineering with the advantages of application frameworks without an extra layer, we aspire a framework, which is geared towards agent systems.

References

1. Agent Oriented Software Pty. Ltd.: JACK Intelligent Agents – WebBot Manual. Agent Oriented Software Pty. Ltd. (Jun 2005), http://www.aosgrp.com/documentation/jack/WebBot_Manual.pdf, (09.07.2011)
2. Ahrndt, S.: HAI – A Human Agent Interface for JIAC. Diploma thesis, Technische Universität Berlin, Berlin, Germany (May 2011)
3. Albayrak, S.: Intelligent Agents in Telecommunications Applications – Basics, Tools, Languages and Application, Frontiers in Artificial Intelligence and Applications, vol. 36. IOS Press, Van Diemenstraat 94, 1013 CN Amsterdam, The Netherlands (1998), iSBN 90 5199 295 5
4. Bellifemine, F., Poggi, A., Rimassa, G.: JADE - a fipa-compliant agent framework. Proceedings of the Fourth Practical Application of Intelligent Agents pp. 97 – 108 (Apr 1999), http://jmvidal.cse.sc.edu/library/jade.pdf, (09.07.2011)
5. Ford, N.: Art of Java Web Development: Struts, Tapestry, Commons, Velocity, JUnit, Axis, Cocoon, InternetBeans, WebWork. Manning Publications (Nov 2003)
6. Hirsch, B., Konnerth, T., Heßler, A.: Merging agents and services – the JIAC agent platform. In: Bordini, R.H., Dastani, M., Dix, J., Amal, E.F.S. (eds.) Multi-Agent Programming: Languages, Tools and Applications, pp. 159–185. Springer Berlin Heidelberg (2009)
7. Howden, N., Rönnquist, R., Hodgson, A., Lucas, A.: JACK intelligent agents – summary of an agent infrastructure. In: 5th International Conference on Autonomous Agents (2001)
8. Kelemen, V.: JADE tutorial – simple example for using the jadegateway class. MTA SZTAKI (Oct 2006), http://jade.tilab.com/doc/tutorials/JadeGateway.pdf, (09.07.2011)
9. Lützenberger, M., Masuch, N., Hirsch, B., Ahrndt, S., Heßler, A., Albayrak, S.: The BDI driver in the service city. In: Tumer, K., Yolum, P., Sonenberg, L., Stone, P. (eds.) Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, Taipei, Taiwan. pp. 1257–1258. International Foundation for Autonomous Agents and Multiagent Systems (May 2011)
10. Myers, B.A., Rosson, M.B.: Survey on user interface programming. In: CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems. pp. 195–202. ACM Press (1992)
11. Pokahr, A., Braubach, L.: The webbridge framework for building web-based agent applications. In: Dastani, M., Segrouchni, A.E.F., Leite, J., P.Torroni (eds.) Languages, Methodologies and Development Tools for Multi-Agent Systems: First International Workshop, Lads 2007, Durham, UK, September 4-6, 2007. Revised Selected Papers. pp. 173—190. Springer-Verlag New York Inc. (2008)
12. Rocher, G., Brown, J.: The Definitive Guide to Grails. Apress Inc, New York, NY, 2 edn. (Jan 2009)
13. Schmidt, M., Többen, H., Keiser, J., Jentsch, O.: 4. abschlussbericht zum projekt MIATA (a management infrastructure for intelligent agent telecommunication applications). DAI Laboratory, Technical University of Berlin (2000)
14. Thiele, A., Konnerth, T., Kaiser, S., Hirsch, B.: Applying JIAC V to real world problems: The MAMS case. In: Braubach, L., van der Hoek, W., Petta, P., Pokahr, A. (eds.) Multiagent System Technologies. Lecture Notes in Artifical Intelligence, vol. 5774/2009, pp. 268–277. Springer (Sept 2009)
15. Vaucher, J., Ncho, A.: JADE tutorial and primer. Universite de Montreal (Sept 2003), http://www.iro.umontreal.ca/~vaucher/Agents/Jade/JadePrimer.html, (09.07.2011)