Implementing Agent Interoperability Between Language-Heterogeneous Platforms

Giacomo Cabri, Elton Domnori, Davide Orlandini
Dipartimento di Ingegneria dell’Informazione
Università di Modena e Reggio Emilia
Via Vignolese, 905 - 41125 Modena - ITALY
{giacomo.cabri, elton.domnori, davide.orlandini}@unimore.it

Abstract

One of the most important features of a MAS (Multi-Agent System) is the cooperation and the coordination among agents placed in distributed environments. Due to the high heterogeneity of agents and platforms, interoperability has become a key issue for the use of MASs in academic and industrial fields. Several approaches to interoperability proposed so far are based on the JAVA technology exploiting the homogeneity of the programming language and technology.

In this paper we introduce a new middleware that enables the interoperability among agents executing in different platforms, implemented with different programming languages. Our approach is FIPA-IEEE compliant and programming-language independent.

1. Introduction

In the last years, the distributed applications based on the software agent technology have gained more and more relevance in both academic and industrial contexts. Several frameworks implementing agent platforms have been made available. Many of them are open source, implemented exploiting the Java programming language, while others are distributed with a commercial license [7].

Up to now, several solutions have been proposed with regard to interoperability between agent platforms, the most wide-spread and exploited one is represented by the FIPA-IEEE specifications [1]. Nevertheless, all the reported results are related to Java platform, relying on homogeneity of programming language and technology [6]. No experience has been reported about interoperability between platforms implemented by different programming languages, and this remarks the difficulties in realizing a solution towards the interoperability between platforms developed exploiting different technologies.

So, we faced this situation and the aim of our work was to concretely realize interoperability between programming-language heterogeneous platforms.

The two most exploited languages for the development of Multi-Agent System are Java and C#. Some of the platforms developed using Java are: MadKit [3], Jadex [2], Dimax [5], and Jade [4], while among platforms developed in C# we can mention AgentService [10], PASIBC [8] and MAPNET [9].

The platforms considered in our work are: JADE [4] and AgentService [10]. JADE is currently one of the most spread, stable and advanced framework for the development of multi-agent applications. It is completely written in Java, and offers advanced solutions for an agent platform. Moreover, JADE strictly implements the FIPA-IEEE specifications. AgentService is a framework for implementing multi-agent applications developed at the l.i.d.o. laboratory of the University of Genova. Similarly to JADE, AgentService is compliant to the FIPA-IEEE specifications. AgentService is an open source agent platform developed completely exploiting Microsoft .NET. This is the reason why we chose AgentService for our study about interoperability.

In our work, we considered three different interoperability directions, following the study presented in [6]:

1. **Execution**, i.e. the capability of executing on a platform an agent created for another platform (for instance, a JADE agent on the AgentService platform);
2. **Mobility**, i.e. the capability of migrating to a platform an agent executing in another platform (for instance, migrating a JADE agent to the AgentService platform);
3. **Inter-agent communication**, i.e. the capability of exchanging messages between agents executing on different platforms.

Our result is that only the inter-agent communication is currently feasible. The other two directions are very hard,
mainly because they involve a programming language translation, but also because the “execution environment” is very different. Nevertheless, in this paper we provide readers with some useful considerations about these two directions.

So, we decided to follow the inter-agent communication direction, studying and implementing a middleware to support it. In this paper we present the features of the developed middleware.

The paper is organized as follows: in the next section related work will be discussed. In Section 3 we discuss the three interoperability directions, pointing out which are feasible and which are not. Then, we will present the developed middleware (Section 4) that enables the communication between heterogeneous platforms. Before the conclusions (Section 6), Section 5 presents an application example we developed.

2. Background

Interoperability between MASs has always drawn the attention of researchers as it represents one of the crux for MAS use. Some of the studies done so far has followed two directions: exploiting homogeneity and creating a common language for communication.

Towards homogeneity we can mention JIMAF [6], a middleware that proposes a high-level approach grounded on the concept of software layering. It is able to provide interoperability of execution, communication and migration. MASs which aim to interoperate must adopt this common solution to let their agents interoperate.

In 1996, the Foundation for Intelligent Physical Agents (FIPA), a forum of international companies with a strong focus in the telecommunication industry, was formed to promote the uptake of software agents in businesses at large. FIPA’s official mission statement was the promotion of technologies and interoperability specifications that facilitate the end-to-end interworking of intelligent agent systems in modern commercial and industrial settings. Since 1997, when FIPA produced its first specifications over an agent communication language (a.k.a. FIPA-ACL), it has become the most used standard in MAS interoperability.

We found such research about the interoperability between heterogeneous MASs as the one we consider in this paper. To face the problem of heterogeneity, in our approach, we will rely on the well-established and widespread standard specification of IEEE-FIPA. These specifications have been adopted by many MASs and are currently both de facto and de jure standard specifications.

3. Agent Interoperability

In this section we provide some considerations about the three interoperability directions we considered in our work, while in the rest of the paper we focus on the third one.

3.1. Execution

Existing middlewares, such as JIMAF [6], allow the execution of agents on different MAoS implemented over the same language. In our case, considering two different languages, the major obstacle is given from the different technology that agents and applications exploit. Besides a difference in the platform architecture and the technology solutions implemented by the different frameworks AgentService and JADE, there is also a relevant difference related to the two different development languages, C# and Java. From their sources different bytecodes are produced, running on different frameworks and not compliant (the .NET Framework and the JVM). Therefore, it is difficult to do the conversion (the so-called “porting”) of the code of agents and to enable them to run on both platforms.

In the case of MAS implemented in different technologies, to enable the execution of an agent on instances of a different platform, the work to be done is essentially a proper “rework” of the code structure. In some cases, different method names in different MASs offer the same functionality and the conversion of the application logic code is not required. Only the exact match of these function names in necessary. In the case of JADE and AgentService, the entire source code must be translated (including the ones related to the application logic), as well as it should be reworked in the format required by the destination platform.

The porting of code from the C# to Java language, and vice versa, is a hard operation that presents several difficulties. The use of appropriate tools such as Net2Java, CS2J, j2ctranslator, j# and JLCA (Java Language Conversion Assistant), is not enough. The translated code must be often manually revised and sometimes, mainly if specific library classes are involved, the translation cannot be performed. This, because of the lack of the specific functionalities of the destination language environment. Further difficulties given from the need to convert whole classes, functions or data structures, inherent in the nature of code, are added to the previous one. In the case we want to translate code that uses classes or data structures specifically implemented for AgentService or JADE (situation that in some cases even involves the building of entire classes and features), readers can realize how the capability to run agents on platforms AgentService and JADE is extremely difficult to obtain. The translation process would lead to good results only in cases of simple fragments of code, which use data types and features common to both languages. The translation of the entire code of agents is much more difficult.

So, we decided to discard this direction and not to include in our middleware any support for the execution of “foreign” agents.
3.2. Mobility

Many of the considerations reported in Section 3.1 about the possibility of translating the code of an agent and the execution on different platforms also apply to the mobility of agents between a platform and another of a different kind. The fact that the code of an agent can not be translated from the language of a platform to a different one, precludes the possibility of mobility at the beginning, because mobility requires that the agent can run on both platforms of origin and destination. If this problem had a solution, it would still need to find an efficient solution to other problems, such as the storage and transmission of the state of the agents. It is important to note that our study has considered only the possibility of weak mobility and, therefore, the term “state” means the data associated with the agent.

A solution for data transmission would require the translation of the data structures from the origin platform into a format compliant with the target platform. Unfortunately, the functionality of serialization and deserialization of objects provided by Java and .Net can not be exploited, because they adopt a different format, not compliant with each other. Some possible solutions are to exploit an XML file formatted according to an appropriate DTD or other file types, whose creation and interpretation must be in charge of an appropriate middleware.

In any case, these solutions are not implemented currently, due to the fact that the impossibility of converting the code precludes the possibility of realizing a middleware for mobility between the two platforms AgentService and JADE.

3.3. Inter-agent Communication

With regard to the communication between agents, we have studied and designed a solution that uses the specifications defined by the IEEE-FIPA communication between agents (called FIPA-ACL). These specifications define the structure of a message, specifying the fields that must or may contain. FIPA-ACL specifications also specify a set of “message types” (so-called “performative type”) which are intended to identify the features for which the message was conceived, and they allow its use in the context of more or less complex conversations. FIPA-ACL specifications also define a set of “communication protocols” which implement different types of conversations (of common use) with defined rules.

The communication between agents was the only aspect in which it is possible to achieve interoperability between AgentService and JADE. For the aspects of Execution and Mobility it was not possible to obtain appreciable concrete results due to obstacles of various kind as reported in sections 3.1 and 3.2.

The results have been obtained through the implementation of a middleware that deals with the communication between two instances of the platforms (one AgentService and the other JADE). Messages compliant to the FIPA-ACL specifications are coded according to a suitable format to be reconstructed and sent to recipient agents on the destination platform. In Section 4 we will present the developed middleware.

4. The developed Middleware

The main idea of this work is to implement a middleware that provides the same functionality for the AgentService and JADE platform.

The realized middleware is composed of two components: the former is intended for the host running AgentService, the latter for the host running JADE. Figure 1 shows a diagram of the middleware. In the right side are reported the modules that are part of the AgentService component, while in the left side are reported the modules running on JADE. The modules are called, respectively, NetExternalRuntime and JavaExtRuntime and are respectively implemented in the form of a Windows Service and a Java application. NetExternalRuntime is a service that is activated and remains running in the context of a Microsoft Windows OS, while JavaExtRuntime consists of a core class (which implements the method main() of the component) and two classes that implement service agents.

The communication between the two components is obtained through the use of two sockets per component; used in unidirectional mode of transmission they cover both directions.

The Windows service NetExternalRuntime is in charge of: receiving incoming messages from the instance of AgentService, encoding them to produce a properly formatted string, and sending it to the component JavaExtRuntime using a socket, which will grant the reconstruction of the message and the delivery to the recipient(s). NetExternalRuntime is also in charge of waiting (on the second socket) for receiving coded messages from JADE. This task is performed by a dedicated thread, and once the string corresponding to the message is received, NetExternalRuntime performs its reconstruction and the delivery to the recipient(s).

JavaExtRuntime and NetExternalRuntime therefore play the same functionality, except in the case of NetExternalRuntime where functions are performed by service methods running on a dedicated thread, while in JavaExtRuntime the same functions are performed in the context of agents by means of appropriate behaviours (which are forced to run on dedicated thread).
In both components, for the communication with instances of the platforms, we have exploited the functionalities made available for access to the platform by external applications (provided respectively by the two frameworks AgentService and JADE), not detailed here due to space limitations.

By means of the introduction of two special performative types called **YELLOW_PAGES_REQUEST** and **YELLOW_PAGES_RESPONSE**, the remote access to the service of the yellow pages of the two MAS has been implemented for both platforms. The service returns a list of local names of the agents registered for a specific service.

Unfortunately, we were able to test only the access to the Yellow Pages service of JADE from agents executing on an AgentService platform. The access to the same service on AgentService by JADE agents was more problematic. One of the reasons is that AgentService is lacking in terms of documentation more than JADE, and the use of some features (particularly when they are implemented in external modules, such as the access to a platform from a remote application) is not as intuitive and well documented as in the case of JADE. Some of the features available as add-on modules are also released in beta version and they are not entirely free from bugs. So, the development and test of the remote yellow pages service for AgentService is delegated to possible future implementation. Instead, the yellow pages service for JADE is fully working. Its implementation can be considered as an implementation model for services that could be realized in the future. Additional services, such as access to the service of white pages, could in future be made with few significant changes to the code created for the yellow pages service.

For the development of agent-based distributed applications we provide an additional library along with the middleware, which implements two classes for creating and reading messages in AgentService compliant to FIPA-ACL. The methods provided by the two classes, called **ACLMessageBuilder** and **ACLMessageReader**, are equivalent to the methods provided by the class **ACLMessage** in JADE and are reported in Table 1 and in Table 2.

---

**Table 1. Methods of the class ACLMessageBuilder**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACLMessageBuilder(AgentMessage msg, int performativeType, Behaviour ownerBehaviour)</td>
<td>Implements methods for creating messages compliant to FIPA-ACL.</td>
</tr>
<tr>
<td>ValidateMsg(AgentMessage msg)</td>
<td>Validates the message structure.</td>
</tr>
<tr>
<td>SetDestinationRuntime()</td>
<td>Sets the destination runtime for the message.</td>
</tr>
<tr>
<td>SetPerfornative(AgentMessage msg, int performativeType)</td>
<td>Sets the performative type of the message.</td>
</tr>
<tr>
<td>SetSender(string sender)</td>
<td>Sets the sender for the message.</td>
</tr>
<tr>
<td>AddReceiver(string receiverLocalName)</td>
<td>Adds a receiver to the message.</td>
</tr>
<tr>
<td>AddReplyTo(string replyToLocalName)</td>
<td>Adds a reply-to address to the message.</td>
</tr>
<tr>
<td>SetContent(string content)</td>
<td>Sets the content of the message.</td>
</tr>
<tr>
<td>SetLanguage(string language)</td>
<td>Sets the language of the message.</td>
</tr>
<tr>
<td>SetEncoding(string encoding)</td>
<td>Sets the encoding of the message.</td>
</tr>
<tr>
<td>SetOntology(string ontology)</td>
<td>Sets the ontology of the message.</td>
</tr>
<tr>
<td>SetProtocol(string protocol)</td>
<td>Sets the protocol of the message.</td>
</tr>
<tr>
<td>SetConversationId(string conversationID)</td>
<td>Sets the conversation ID of the message.</td>
</tr>
<tr>
<td>SetReplyWith(string expression)</td>
<td>Sets the expression for the reply of the message.</td>
</tr>
<tr>
<td>SetInReplyTo(string expression)</td>
<td>Sets the expression for the in-reply-to address of the message.</td>
</tr>
<tr>
<td>SetReplyBy(DateTime timeLimit)</td>
<td>Sets the time limit for the reply of the message.</td>
</tr>
</tbody>
</table>

---
5. An Application Example

To test the proposed middleware, a sample application has been designed and developed. The application implements a service for the sale of CDs, identified by title and artist. The service is implemented with a variable number of agents residing on an instance of the JADE platform which act as sellers (Seller). Each seller offers a list of CDs stored in a catalogue implemented by a data structure of type Hashtable. To each CD, in the catalogue, is associated its price in Euro. When started, each seller agent displays a GUI that allows the inclusion of a CD in the catalogue along with its price. The client side application is represented by a set of buyers (Buyer) implemented by agents residing on an instance of the AgentService platform.

To these buyer agents can connect one or more external applications that represent different buyers (Buyer). The applications representing buyers are Windows Forms applications that use the remote access service made available by the AgentService platform to communicate with buyer agents, and to forward requests in order to purchase CDs. The communication between buyer agents and seller agents implements a conversation protocol similar to the contract-net one [1].

Buyer agents have access to a list of registered vendors on the JADE platform, which is updated at regular intervals of sixty seconds, by means of a remote request to the JADE yellow pages service, implemented by the middleware. The seller agents are in fact registered for the service called “CD-seller” in the Directory Facilitator of JADE. Upon receiving a purchase request from a (human or application) buyer, a buyer agent forwards a call for proposal message to seller agents that it has the reference to. The latter reply the buyer agent with a propose message, if they can provide a copy of the CD, or with a refuse message otherwise. The buyer agent evaluates the proposals received to decide which one is the most convenient, and then it sends an accept-proposal message to the seller agent that has submitted the best proposal.

Finally, the seller agent that has received the acceptance for the proposal sends an inform message to the buyer, to inform the buyer agent about the transaction. The transaction confirmation is also sent to the application of the human buyer, together with the name of the buyer agent and the purchase price.

The application has been tested by simulating various purchase transactions, with a variable number of agents registered on the two platforms, run in the context of two different hosts. The test has reported a correct work of the agents, and the middleware has allowed the communication between the two platforms without significant delays in the transmission of messages. During the test new seller agents were activated dynamically, to verify the updating of the seller list made available to buyer agents through the access service to yellow pages implemented by the middleware, and even in this case, the application responded correctly. In Figure 2 a diagram of the application is reported, which shows, in addition to the middleware components, also

Table 2. Methods of the class ACLMessageReader

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>string GetPerformative(AgentMessage msg)</td>
</tr>
<tr>
<td>string GetSender(AgentMessage msg)</td>
</tr>
<tr>
<td>string[] GetReceivers(AgentMessage msg)</td>
</tr>
<tr>
<td>string[] GetReplyTo(AgentMessage msg)</td>
</tr>
<tr>
<td>string GetContent(AgentMessage msg)</td>
</tr>
<tr>
<td>string GetLanguage(AgentMessage msg)</td>
</tr>
<tr>
<td>string GetEncoding(AgentMessage msg)</td>
</tr>
<tr>
<td>string GetOntology(AgentMessage msg)</td>
</tr>
<tr>
<td>string GetProtocol(AgentMessage msg)</td>
</tr>
<tr>
<td>string GetConversationID(AgentMessage msg)</td>
</tr>
<tr>
<td>string GetReplyWith(AgentMessage msg)</td>
</tr>
<tr>
<td>string GetInReplyTo(AgentMessage msg)</td>
</tr>
<tr>
<td>DateTime GetReplyBy(AgentMessage msg)</td>
</tr>
</tbody>
</table>

With regard to communication with JADE agents, any additional library was not deliberately introduced (currently). The intention was to maintain a non-invasive approach against existing JADE applications, in order to use them with the implemented middleware without changes to application code (the so-called Low-Level modifications [6]). Only if the agents executing on JADE want to start a conversation with a remote platform, some attentions must be paid in the development phase. In this case it is necessary that the local names of remote recipients are specified in a special user-defined field of the message, called externalReceivers.

It should be pointed out that in the current state of implementation, some solutions, such as the management of conversations, are not optimal and can be replaced with better solutions in the future. In the current management of conversations, we keep track of the recipients of any replies to a message by including their local name (in a suitable format), within the field conversation-id. This solution is not elegant and could be replaced by better solutions, such as solutions that exploit data structures of type Hashtable to keep references to the recipients of any reply. A further improvement would be to use data structures of type Hashtable along with communication protocols for the management of conversations. The implementation of the FIPA standard communication protocols would also be a further important aspect with regard to the interoperability between the AgentService and JADE platforms.

the entities that implement the service for the sale of CDs.

6. Conclusions

In this paper we have presented a middleware to enable the interoperability between agents execution in different platforms. While in the past some approaches have been proposed for platforms implemented in the same language [6], the innovation of our work is that we have considered two platforms written in different languages. In particular, we considered JADE (in Java) and AgentService (in C#). Our approach can be easily applied also to other heterogeneous platforms compliant to the FIPA specifications.

The presented infrastructure is designed to allow easy implementation, in appropriate forms, of additional features or alternative solutions to those currently adopted. Among the features currently implemented is to highlight the remote access to the yellow pages service of a platform. An important consideration is about the FIPA specifications. Our work has remarked that their concrete exploitation enables the interaction between agent platforms different also for implementation language.

In this paper we have also discussed about the portability and the mobility of agents between platforms implemented in different languages, concluding that this is currently not possible, not only for problems related to the portability of the implementation programming languages, but also because of the different environment provided by different platforms. With regard to future work, among the features that can be implemented in the future we can mention a service of white pages and the implementation of the FIPA standard communication protocols for conversations. Some solutions currently implemented, such as the management of the target in complex conversations, in the future can be optimized and made more elegant and efficient.

References