Runnable specifications of interactions for open multi-agent systems

Philippe Mathieu, Jean-Christophe Routier and Yann Secq
LABORATOIRE D’INFORMATIQUE FONDAMENTALE DE LILLE, UMR CNRS 8022,
UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE, FRANCE
Email: {Philippe.Mathieu, Jean-Christophe.Routier, Yann.Secq}@lifl.fr

Abstract—Designing distributed systems is hard, designing open distributed systems is even harder. Several approaches are competing: from object oriented technologies and Web Services to agent oriented technologies and Agent Communication Languages. We think that these approaches should be thought as complementary and useful at different levels. In this paper, our work is focused on the notion of interaction, that we believe is the key to ease (and even enable) interoperability within open distributed systems. We propose a formalism to describe a bird eye view of multiple role interactions, that can be translated to runnable objects for each participant, thanks to code generation. Then, we propose the RIO methodology for the design of open multi-agent systems based on an engineering of interaction protocols. These interaction protocols are described in term of conversation between micro-roles characterized by their skills. Micro-roles are then gathered in composite roles to build abstract agents. Finally, these latter can be dynamically distributed on running agents of a multi-agent system.

I. INTRODUCTION

The quest for interoperability has bring to computer science important standards that have enabled greater interaction and integration between software systems. From encoding standards like the venerable ASCII to the new XML-based technologies, the development of standards, i.e. a bodies of agreed conventions, has slowly driven the whole computer industry.

The last decade saw the birth of interoperability on the functional level, with the works done around CORBA. This common object request broker architecture has provided to object-oriented systems a greater level of abstractions through the use of the IDL language to define data structures and object interfaces, and has enabled interoperability between several object-oriented languages.

On another side of research, the idea of an agent based software engineering has appeared roughly ten years ago, with the paper from Shoham entitled Agent Oriented Programming[1]. Since these days, several methodologies have been proposed to help developers in their analysis and design[2], [3]. For that, the concepts of role, interaction and organization are often proposed to facilitate the decomposition and the description of distributed systems.

However, we think that suggested methodologies do not clearly identify the various levels of abstraction making it possible to break up a system and especially they generally do not propose pragmatic concepts or principles facilitating the realization of such systems. Particularly, the dream of agents interoperability through the use of high level message passing (i.e. Agent Communication Languages) has failed to deliver a language or platform demonstrating its value. Nevertheless, the FIPA organization is slowly standardizing some aspects of agent technologies, and some interoperability challenges are beginning to bring the lowest level of interoperability (transport and basic services like registration and agent and service lookup).

We believe that this is not sufficient and that standardization should be concentrated on agent interactions. The conversations that agents can handle should be formalized, so computers could manipulate them and their semantic would have to be sharp. If we look some of the FIPA specifications on interaction protocols like the FIPA Contract Net, we must conclude that the semantic of this interaction is loosely defined and requires the interpretation of agent developers to be integrated within their platform.

Thus, the introduction of a level of interaction management while rigidifying in a certain way the possible interactions between agents, brings guarantees on coordination and allows the reification of these interactions. The table below, inspired by work of Singh[4], illustrates the various levels of abstractions within a multi-agent system:

<table>
<thead>
<tr>
<th>Applicative skills</th>
<th>Business knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent models and system skills</td>
<td>Agent oriented design</td>
</tr>
<tr>
<td>Conversation management</td>
<td>Interaction oriented design</td>
</tr>
<tr>
<td>Message transport</td>
<td>Agent platform (i.e. agents container)</td>
</tr>
</tbody>
</table>

The FIPA organization has already proposed standards for the lowest level. We believe that the next major achievement will be on interaction formalization and automation (an analogy could be done with the hype around web services choreography). Our work is very close to the paradigm called Interaction Oriented Programming, that has been defined by Singh[4]: We introduce interaction-oriented programming (IOP) as an approach to orchestrate the interactions among agents. IOP is more tractable and practical than general agent programming, especially in settings such as open information environments, where the internal details of autonomously developed agents are not available. It is the point of view that we adopt, by proposing a pragmatic method

1Reification is the process by which a user program or any aspect of a programming language, which were implicit in the translated program and the run-time system, are brought to the fore using a representation (data structures, procedures, etc.) expressed in the language itself and made available to the program, which can inspect them as ordinary data.
for the design and realization of multi-agent systems, relying on the concept of *executable* specification of interaction protocols. Thus, our proposal relies on a model of generic minimal agent and a model of executable specification of interactions. The agent model is the infrastructure allowing the deployment and the management of interactions, while the specification of the interactions describes a global sight of the conversations between the agents of the system.

In the first part of this article, we introduce the RIO methodology that relies on an engineering of interaction protocols to define multi-agent systems (ie. complex distributed systems). This methodology emphasizes the formalization of interaction protocols and the use of the notion of role and organization to analyze and specify a multi-agent system. Then, the second part presents the core piece of the RIO methodology : the formalism used to describe and realize interaction protocols and its associated generic agent model to support the management of these protocols. The last part of this paper describe related works, both on agent oriented methodologies and interaction-based systems.

### II. RIO : Roles, Interactions and Organizations

In this section, we will present the RIO methodology which relies on an engineering of interaction protocols. This methodology falls under the line of GAIA[5], and thus aims the same applicability. However, GAIA remains too general to easily be able to go from the system design stage to its realization. The purpose of our proposal is to facilitate this transition.

The RIO methodology relies on four stages. The two first represent reusable specifications, while the two last are singular with the application (figure 1). Moreover, we will not speak about *application*, but about an agent society. Indeed, the RIO methodology proposes an incremental and interactive construction of multi-agent systems. By analogy with our generic minimal agent model (section III), where an agent is a container which can receive skills, we see a multi-agent system like a container that has to be enriched by interactions. We will detail this approach by studying the four stages of our methodology.

#### A. Specification of interaction protocols

The first stage consists in identifying the involved interactions and roles. Then, it is necessary to determine the granularity of these interactions. Indeed, for reasons of re-use of existing protocols, it is significant to find a balance between protocols using too many roles, these protocols becoming too specific, and protocols where there are only two roles, in this case the view of the interaction is no more global.

The specification of the interaction protocols can then be done in three ways : either *ex-nihilo*, by specialization, or by composition. Creation *ex-nihilo* consists in specifying the interaction protocol by detailing its cartouche (figure 4). Specialization makes it possible to annotate an existing cartouche. Thus, it is possible to specify the cartouche of an interaction protocol such as FIPA CONTRACTNET, its specialization will consist in changing micro-roles names to adapt them to the application, to refine message patterns, and if required to modify insertions/extractions of information. Finally, the composition consists in assembling existing protocols by specifying associations of micro-roles and information transfers. At the end of this stage, the designer has a set of interaction protocols. He can then pass to the description of composite roles, which will allow the aggregation of micro-roles that are involved in complementary interactions.

#### B. Aggregation of micro-roles to build composite roles

This second stage specifies role models. These models are abstract reusable descriptions. The composite roles correspond to a logical gathering of micro-roles. These patterns define *abstract* roles, which gather a set of consistent interaction protocols. For example, a composite role SUPPLIES MANAGEMENT will gather the micro-role BUYER within the PROVIDERS SEEKING interaction protocol and the micro-role STOREKEEPER of the interaction SUPPLIES DELIVERY. Indeed, a role is generally composed of a set of tasks which
can be, or which must be carried out by the agent playing this role. Each one of these tasks can be broken up and be designed as a set of interactions with other roles. The concept of composite role is thus used to give a logical coherence between the micro-role representing the many facets of a role.

C. Specification of the agent society

This third stage can be regarded as a specification of an abstract agent society, i.e. a description of abstract agents and their occurrence, as well as the link between composite roles and organizations. Once the set of composite roles is created, it is possible to define the abstract agents (patterns of agent, or agent template), which are defined by a set of composite roles. These abstract agents describe applicable agent models. These models are specific, because they introduce strong dependencies between composite roles.

For example, the Office Stationery Delivery composite role could be associated with the Traveling Expenses Management composite role to characterize a Laboratory Secretary abstract agent (fig 3). Once abstract agents are defined, it is necessary to specify their occurrence in the system. It means that each abstract agent has an associated cardinality constraint that specifies the number of instances that could be created in the system (exactly N agents, 1 or more, *, or [ m..n ]). The second part of this stage consists in specifying for each abstract agent, and even for the composite roles of these agents, which organization should be used to find their acquaintances. Indeed, when agents are running, they have to initiate interaction protocols, but in order to do that they initially have to find their interlocutors. The organization is used as a media for this search. This association makes it possible to use various organizations for each interaction protocol.

D. Deploying specifications within a multi-agent system

This last stage specifies deployment rules of the abstract roles on the running agents of a system. We have a complete specification of the agent society, that can be mapped on the concrete agents of the multi-agent system. For that, it is necessary to indicate the assignments from abstract agents to concrete ones. Then, the connection between a skill interface and its implementation is carried out. The designer indeed must, according to criteria that are specific to the hosting platform or applicable, bind the implementation with skill interfaces.

It is during deployment that the generic agent model is justified as a support to dynamic acquisition of new skills related with the interaction. Indeed, the interaction, once transformed by the projection mechanism, is represented for each micro-role by a Colored Petri Net and its associated skills. All these information are sent to the agent, which adds applicable skills and delegates the CPN to the conversation manager. When an agent receives a message, the conversation manager checks if this message is part of a conversation in...
progress (thanks to the conversation identifier included in the message), if it is the case, it delegates the message processing to the concerned CPN, if not he seeks the message pattern matching the received message and instantiates the associated CPN. If it does not find any, the message will have to be treated by the agent model. We will now detail the interaction protocol formalism and its translation in runnable objects.

III. AN ENGINEERING OF INTERACTION PROTOCOLS

The heart of our proposal is a formal model to describe interaction protocols, and a transformation mechanism to generate the code that is necessary to the management of these protocols. In order for running agents to be able to exploit these new interactions, we rely on a minimal generic agent model, which authorizes the incremental construction of agent per skills addition. Thus, we will initially present this generic agent model, then we will study the model of specification of interaction protocols and the associated transformation mechanism.

A. Our minimal generic agent model

The basis of our model is on the one hand the interactive creation of agent, and on the other hand a search on the fundamental functionalities of agenthood. We are not interested in the description of the individual behavior of agents, but rather in the identification of functions that are sufficient and necessary to an agent. Indeed, the management of interactions, the knowledge management or the management of organizations, are not related to the agent model, but are intrinsic characteristics with the concept of agent.

In our model, an agent is a container which can host skills. A skill is a coherent set of functionalities accessible through a neutral interface. This concept of skill is to be brought closer to the concept of software component in object oriented technologies. Thus, an agent consists of a set of skills which carries out various parts of its behavior. We identified four layers which are characterized by the various levels of abstraction of functionalities that are proposed (table 1).

The first level corresponds to “system” skills, i.e. the minimal functionalities allowing to bootstrap an agent: the communication (emission/reception of messages) and the management of skills (dynamic acquisition/withdrawal of skills). The second level identifies agent skills: the knowledge base, media of interaction between skills and the place of knowledge representation, the management of interaction protocols (cf. following section) and the management of organizations (cf. last section). The third level is related to skills that define the agent model (reactive, BDI...), while the last level represents purely applicative skills. Rather than skills carrying out these various levels, it is the functionalities that they represent which are fundamental: the management of the communications, just like the knowledge base can be implemented in different ways, but it is necessary to have these functions within the agent. Thus, the first and the second level characterize our generic minimal agent model. This model is generic with respect to the agent models that can be used, and minimal in the sense that it is not possible to withdraw one of the functionalities without losing a fundamental aspect of agenthood.

A skill is made of two parts: its interface and its implementation. The interface specifies the incoming and outgoing messages, while the implementation carries out the processing of these messages. This separation uncouples the specification from its realization, and thus makes it possible to have several implementations for a given interface. The interface of a skill is defined by a set of message patterns which it accepts and produces. These messages must be discriminated, it is thus necessary to type them.

\[
\text{interface} := ((m_\text{in})^+, (m_\text{out})^*)^* \\
\text{where } m_\text{in} = \text{message pattern}
\]

The typing of message patterns can take several forms: a strong typing, which has the advantage of totally specifying the interfaces, while a weak typing offers more flexibility with regard to the interface evolution. Thus, if the content of messages are expressed in KIF or DAML+OIL, a strong typing will consist of an entire message checking, while a weak typing will only check it partially.

B. A runnable specification of interaction protocols

Many works have been done to specify interaction protocols. Recently, AgentUML[7] was defined like an extension of UML, to specify the conversations between agents, in particular by specializing sequence diagrams in UML. However, these specifications require the interpretation of developers, which must then translate them in their own system. Works of Labrou and Finin[8] explore the use of Colored Petri Nets[9] (CPN) to model conversations between agents. In [10], the same approach is used, but the concept of Recursive Colored Petri Nets is introduced to support conversations composition. Our work follows the same principles: to represent interactions in a global way, and to use a recognized and established formalism. However, contrary to preceding works, our goal is to produce an executable specification, i.e., a description of the interaction protocol which can be then directly integrated in a running system. Moreover, CPN are unquestionably adapted to the modeling of concurrent processes, and provide an interesting graphic formalism, but they are unfortunately not really user friendly. This is why it appears preferable to us to define a language adapted to the modeling of interaction protocols, and to use a projection mechanism that translates this language into CPN.

<table>
<thead>
<tr>
<th>Table I</th>
<th>THE FOUR LAYER OF OUR ABSTRACT AGENT MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Applicative skills</td>
</tr>
<tr>
<td></td>
<td>Database access, graphical user interface...</td>
</tr>
<tr>
<td>3</td>
<td>Agent model related skills</td>
</tr>
<tr>
<td></td>
<td>inference engine, behavioral engine, ...</td>
</tr>
<tr>
<td>2</td>
<td>Agenthood skills</td>
</tr>
<tr>
<td></td>
<td>Knowledge base, conversation management, organizations management</td>
</tr>
<tr>
<td>1</td>
<td>Minimal system skills</td>
</tr>
<tr>
<td></td>
<td>Communication and skill management</td>
</tr>
</tbody>
</table>

TABLE I
C. The specification of interaction protocol

The purpose of this model is to ease the specification, the checking and the deployment of interaction protocols within multi-agent systems. On all these stages, the designer has to define the specification, the other stages being automated. For that, we define a formalism representing the global view of an interaction protocol, and a projection mechanism which transforms this global view into a set of local views dedicated to each role. We will initially describe the specification of the global view, before presenting the projection mechanism which generate local views that agents use while a protocol is running. The interaction protocols are regarded here as social laws within the meaning of Shoham[11], that means that agents lose part of their autonomy (conversational rules are static), but the system gains in determinism and in reliability.

Our model relies on the concept of skill, micro-role and a graph that represents the state of the conversation. An interaction protocol formally specifies the course of a conversation (regarded as a social law) between various entities, i.e. the nature of exchanged messages, the flow of these messages and skills that entities must implement for each stage of the conversation. These entities correspond to micro-roles, and are characterized by their name and their skills. One uses a graph to represent the course of the conversation: the nodes represent micro-roles which can be associated to a skill interface, and the arcs with a sending of message between two micro-roles (typed by a message pattern).

An interaction protocol is thus defined by the following elements (figure 4) : the name of the interaction protocol, a textual description of the goal of this protocol, the list of the micro-roles involved in the interaction and the geometrical symbol which is associated to them, ontologies of exchanged messages, a list of information necessary in input and produced at output, an interaction graph gathering information like the temporal course of the conversation, the synchronization and the nature of the exchanged messages, and the skills that are used by micro-roles.

The figure 5 illustrates the interaction protocol for the “Order mission emission” interaction. This interaction involves three micro-roles (Secretary, Director and President), one skill interface (Agreement) and three message patterns (expressed here in terms of XPath expressions, but could also be DAML+OIL expressions).

What it is significant to understand is that the designer has a global view of the interaction: the flow of the messages, their nature (the type of these messages corresponds to the annotations attached to the arcs), needed skills and information used or produced. In addition, all information necessary to the management of the interaction protocol is centralized here and can be used to carry out the generation of the code required to manage this interaction for each micro-role. The designer has thus only to define the interaction protocol by using a graphical tool, the projection mechanism takes care of the generation of descriptions for each micro-role, and the generic agent model can then use these descriptions.
D. Projection mechanism

The preceding section described the formalism representing interaction protocols, we will now explain the transformation making it possible to obtain a runnable specification. The specification of interaction protocols gives to the designer a global view of the interaction. Our objective is to generate for each micro-role a local view starting from this global view, this one could then be distributed dynamically to the agents of the system. The projection mechanism transforms the specification into a set of automata. More precisely, an automata is created for each micro-role. This automata manages the course of the protocol: coherence of the protocol (messages scheduling), messages types, side effects (skill invocation). The implementation of this mechanism is carried out by the generation of Colored Petri Nets. Indeed, we use the color of tokens to represent messages patterns, in addition we have a library facilitating the interactions between generated networks and the agent skills. On the basis of the interaction graph, we create a description of Colored Petri Net for each micro-role, and we transform this textual description to a Java class. This class is then integrated within a skill, which is used by conversation manager skill (level 2 in table I). The interest of this approach is that the designer graphically specifies the global view of the interaction, the projection mechanism generates the skill needed to the management of this interaction. Moreover, thanks to the dynamic skill acquisition, it is possible to add new interaction protocols to running agents of the system.

E. Knowledge and organization management

The knowledge management and the management of organizations are also mandatory functionalities of agent, and they should not be enclosed within the agent model. The knowledge management gathers at the same time their representation, the information storage, the means of reaching and of handling them. The implementation of these functionalities is strongly dependent on the used agent model (third level of table I). The concept of organization is necessary to structure interactions that intervene between entities of the system. This concept brings some significant benefits: a means to logically organize the agents, a communication network per defect and a media to locate agents, roles or skills. Moreover, its reification provides a door in the system, making it possible to visualize and to improve interactions between agents[12].

IV. RELATED WORKS

Lots of works have been done on agent technologies. These works can be split in two approaches: high level methodologies and specialized pragmatic tools. The first category is illustrated by the works done on multi-agent methodologies, which provide conceptual frameworks to analyze such systems. The second category is made of several tools that have been developed to facilitate the implementation of such systems and thus concentrate on specific aspects of multi-agent systems. In this category, we choose to be focused on tools that tackle the complexity of interaction design.

A. Agent oriented methodologies

Several agent oriented methodologies have been proposed like AALAADIN[13] or GAIA[5]. It is significant to notice that these methodologies do not make any assumption on agent models and concentrate mainly on the decomposition in term of roles of a complex system. This point is fundamental, in particular because of the multiplicity of available agent and multi-agent systems models. These methodologies are interesting on many points, but remains too general to bridge the gap between the transition from the design stage to its concrete realization. Moreover, the various levels of communication are not clarified in the description of interaction protocols. Indeed, works on agent communication languages (ACL) identify three levels that constitutes a conversation: the semantic, the intention (these two are expressed through languages like KIF or SL, and KQML or FIPA-ACL), and the interaction level. However, even by considering heterogeneous platforms sharing the same ontology, it remains difficult to have guarantees on the respect of interaction protocols. Particularly, in GAIA the description of interactions are not
formalized, and it is therefore impossible to have guarantees on the course of a conversation or to allow interoperability between heterogeneous platforms.

B. Interaction languages

To illustrate these approaches based on a formalization of the interactions, we studied three of them: APRIL[14], AGENTALK[15] and COOL[16]. APRIL, which results from works on actor languages[17], is a symbolic language designed to handle concurrent processes, that eases the creation of interaction protocols. In APRIL, the developer must design a set of handlers which treats each message matching a given pattern. According to the same principles, AGENTALK adds the possibility to create new protocols by specialization of existing protocols, by relying on a subclassing mechanism. Another fundamental contribution of AGENTALK is the explicit description of the protocol: the conversation is represented by a set of states and a set of transition rules. The same principle was employed in COOL, which proposes to model a conversation using an finite state automata. In COOL, the need for the introduction of conventions between agents to support coordination is proposed. This concept of convention must be brought closer to the works of Shoham on social rules[11] and their contributions on the global performance of the system.

V. Conclusion

Interoperability is a key component for the development and the integration of multi-agent technologies. Even if the Fipa is working towards this aim, we believe that the main point has not yet be achieved: providing formal descriptions of agent conversations and technological bindings (in the spirit of PIM and PSM within MDA). In this paper, we have presented the RTO methodology that mainly rely on the concepts of the interaction oriented programming. The core of the our methodology is the engineering of interaction protocols, and more precisely the engineering of runnable specifications.

For that purpose, we use a tool facilitating the graphical design of interaction protocols, and a projection mechanism that generates the code corresponding to the vision that each participant has of the interaction. By using these specifications, it is possible to create abstractions characterizing the various roles and agents of a multi-agent system: the composite roles, which gather a set of micro-roles, and abstract agents, which gather a set of composite roles. Finally, the organization is used as a default message routing systems and also a default distributed yellow/white pages scheme.

A framework that implements this approach is under development. Within this framework, agents are composed of skills, which are implemented by OSGi components[18] and described by DAML+OIL ontologies. Interaction protocols are serialized as XML documents that are translated to Java objects that are Colored Petri Nets.

REFERENCES