

Abstracting Communication in Distributed Agent-Based Systems

Monique Calisti
Whitestein Technologies AG
Ghotthardstrasse 50, 8002 Zurich, Switzerland
Voice: +41-1-205 5500 Fax: +41-1-205 5509
E-mail: mca@whitestein.com

Abstract

This paper discusses abstraction components for agent based communication. We report back on our experience in deploying such components for developing agent based solutions and we draw some conclusions on the main benefits and challenges of communication abstractions for multi-agent systems. We finally argue that some of these abstractions have the potential to be re-used or integrated on top of traditional (non-agent) distributed systems.

1 Introduction

Nowadays, communication between distributed software entities populating electronic systems represents a crucial aspect for many applications requiring flexible interactions of distinct and possibly heterogeneous components. This is particularly critical in agent-based systems where the conceptual approach behind any kind of solution design and development strongly relies upon the notion of *interaction* (or social behaviour) of autonomous processes that dynamically coordinate their actions by *communicating* with each other. This paper has the twofold objective of:

- Describing the key abstract components that are currently used in the agent world to design and thereby implement communications between software agents.
- Discussing the practical experience of deploying such communication abstractions when developing agent-based solutions, such as service provisioning in communication networks and e-financing.

This aims, on one side, to identify the main benefits and challenges of communication abstractions for agent-based systems. On the other side, this hopes to stimulate discussion about which agent-tailored communication components may be potentially re-used or integrated on top of existing traditional non-agent based frameworks.

In agent systems, the most commonly adopted approach for communication builds on top of a multi-layered infrastructure that enables decoupling of low-level data transport issues from high-level semantic interoperability aspects. Usually, the transport layer consists of basic building blocks responsible for transparently routing and delivering agent messages to the final proper recipient/s. *Messages* are, in this context, structured data units modelled through the use of a high-level declarative agent communication language (ACL). Therefore, on top of the transport infrastructure, agents interoperate by parsing and interpreting messages in the context of on-going conversations (see Section 2). In this paper, the focus is not on transport low-level issues, but rather on the abstraction components used at the higher level, in order to guarantee meaningful agent communications (see Section 3). This firstly requires a review of the main concepts upon which multi-agent systems are defined (see Section 2). Practical experience in developing agent-based solutions in several domains leads then to the discussion of the main benefits and challenges when deploying abstraction components for agent communication (see Section 4). This also enables to draw some connections with existing frameworks supporting communication of distributed software components in non-agent systems.

2 The Software Agent-Oriented Approach

Today, agent technology represents one of the most dynamic fields in which various techniques are used to build distributed systems with intelligent local components designed to both cooperate and coordinate their activities [7]. Without entering philosophical debates about a formal definition, we assume that agents are software entities [13]¹:

- Performing autonomously (i.e., without the need for direct intervention from humans or other software processes),
- Reacting in time to modifications that happen in the environment they are embedded in,
- Analysing and monitoring the environment permanently,
- Acting before the environment goes to an undesired state in order to meet their design objectives (*proactive* and *goal-driven* behaviour).

Agents receive inputs through sensors and they act on the environment through effectors. Moreover, they exhibit autonomous, reactive, opportunistic and goal-directed behaviour (i.e., the action to perform has to be consistent with the agent's goals). This approach has been adopted for dealing with many tasks in different domains, such as resource allocation, network management, e-commerce, health care, etc. [7], for its intrinsic capability of representing the decentralised nature of many problems, the existence of multiple control and logic components with distinct roles, the multiple capabilities/roles of distributed and eventually self-interested entities. However, for an effective use of this technology and for exploiting typical agents' characteristics such as *social ability*, *responsiveness* and *pro-activity*, a key aspect is the capability of supporting effective agent-to-agent communications.

2.1 Communication in Multi-Agent Systems

In order to achieve their individual objectives or dynamically coordinate their actions, software agents need to interact with each other. *Communication* is at the basis of agents interactions ranging from the ability to exchange comprehensible messages (semantic interoperation), through traditional requests that a particular action is performed (i.e., client-server approach), to the ability of dynamically coordinating and/or negotiating their behaviour.

As anticipated in Section 1, our focus is not on the technical issues related to the transport and delivery of messages (or communicative acts) between distributed agents (such as for instance the selection of the specific data transport protocol, like IIOP, HTTP, SOAP), but rather on the elements that enable a common understanding of the meaning of the tokens of the adopted ACL. A generic communication stack (CS) for agent communications is shown in Figure 1 and logically corresponds to the *abstract communication architecture* standardised by FIPA² in [14]. For full interoperability, at every level of the CS, there has to be a common agreement about the deployed formalism in terms of both syntax and semantics approach. This is where standard methods play the most crucial role. For the different CS layers, FIPA specifies standard components that can be used:

- At the *conversation* level, to determine the valid sequences of messages regulating specific interactions. These valid sequences are called *interaction protocols* (INP) and they have to be known by all the interacting entities (i.e., transmitter and receiver/s).
- At the *message* or *communicative act* level, to make use of a shared ACL, which enables the attitudes regarding the content of the exchange to be expressed. The ACL structure establishes, for instance, whether the content of the communication is, for instance, an assertion, a request or some form of query. The two most known and deployed ACLs in the agent world are FIPA-ACL [17] and KQML [5].
- At the *content expression* level, to describe the content of the message (i.e., a description of a partial world state), which can contain references to *objects*, *actions* and *functions* in a given domain. Existing content languages (CLs) used for this purpose are for instance FIPA-SL, KIF, etc.
- At the *ontology* level, to identify a common approach to define a vocabulary and a set of agreed definitions to describe a specific domain in terms of objects, actions and functions.

¹Other popular formal definitions for agency can be found in [12] and [10].

²FIPA is a non-profit standardisation group that promotes interoperability of emerging agent-based applications, services and equipments (<http://www.fipa.org>).

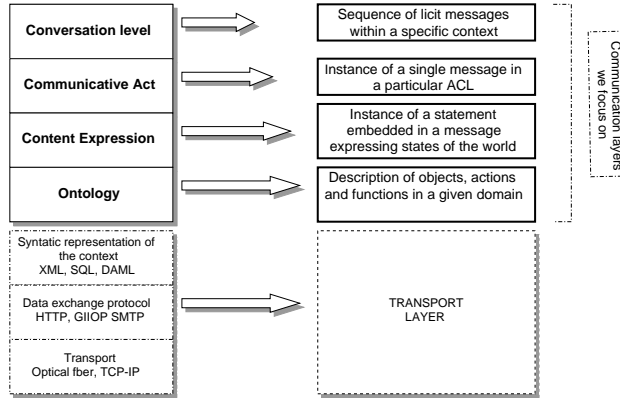


Figure 1: A generic communication stack for agent communications.

A specific example enabling a better understanding of the different communication components listed above, is discussed in the following section. In this context, the term *component* is used to indicate an interface for which the offered and required services/functionalities are well defined.

3 Abstraction Components for Agents Communication

Abstraction methods³ have been extensively proposed as promising techniques to reduce the complexity of problem solving, knowledge representation and systems design. In this latter case, the main idea is to build abstractions that effectively structure the system and map its current structure into a simpler one so that reasoning in this *abstract space* is less complex than in the original one. This has the potential to enable a more effective approach to the definition of the basic building blocks (components) upon which flexible communication of distinct software components can be built [1], [3]. In particular, as observed in [8], “the most powerful abstractions are those that minimise the semantic gap between the units of analysis that are intuitively used to conceptualise the problem and the constructs present in the solution paradigm”. Here, the “problem” we focus on consists of defining the specific sub-system components (or layers in the CS depicted in Figure 1) that can be used for modelling and supporting communication between agents that have possibly been implemented by different developers in distinct environments. The critical issue is then to evaluate how powerful these abstractions are when considering their deployment for the implementation of concrete solutions (see Section 4).

In order to describe the most commonly used abstract components for agent communications, we consider two distinct agents: agent *A* acts on behalf of a final end user willing to make a reservation for a hotel room in Malaga. Agent *B* represents a travel agency that offers various services including hotels reservation.

Interaction Protocols for Agents Conversations. For coordinated interactions, ongoing agent conversations need to fall into typical patterns. This means that certain message sequences are expected, and, at any point in the conversation, other messages are expected to follow. These typical patterns of message exchange are called interaction protocols and they represent the highest abstraction component in the agent CS. In our example, when agent *A* contacts agent *B* requesting the reservation of a hotel room the standard fipa-request protocol [15] is used. This choice identifies which specific kind of ACL messages the agents are expected to exchange (i.e., the valid performatives are *request*, *refuse*, *not-understood*, *failure*, *inform*) and in which order (see Figure 2). The “action” agent *A* is asking to perform is in this case is “booking-hotel”. The specified action together with the additional parameters (e.g., the details about the required room characteristics, the pre-fixed period) represent the *content expression* of the communicative act *request* (see Figure 3).

Agent Communication Language and Content Expression. Once the valid sequence of possible communicative acts is known, it is necessary that agents parse and interpret every single message they receive. This requires the adoption of a standard ACL that has a precisely defined syntax and semantics. At the syntactic level, it is necessary to define the structure of every message, i.e., the various sub-fields that

³See [6] for a more formal definitions of abstraction notions.

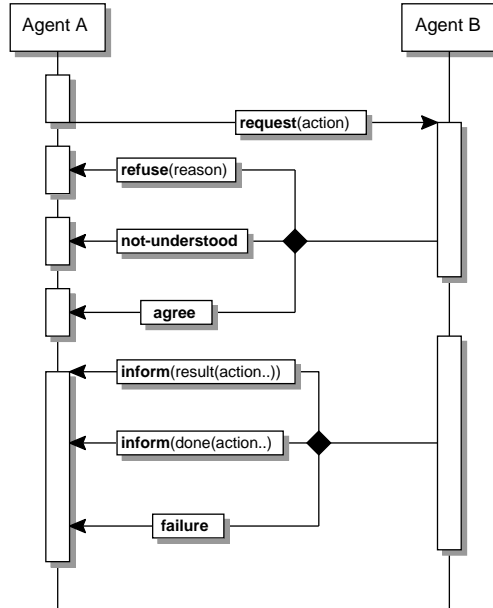


Figure 2: *The standard fipa-request interaction protocol.*

identify the different “pieces” of information embedded on a message. A FIPA-ACL message, for instance, can contain elements such as the *name* of the performative (i.e., the *type* of the communicative act - *request* in our example) and the *sender* and the *receiver* fields denoting the identity of the sender and of the intended recipient/s of the message. The *content* of the message represents the domain dependent component of the communication: the *language* field specifies which content language (CL) is used to express it (FIPA-CL in the example) and the *ontology* field denotes the ontology used to give a meaning to the symbols in the content expression. The *protocol* field specifies the INP that the sending agent is employing with this ACL message.

At the semantics level, the main idea behind the use of an ACL is to capture for every type of the message the mental state of an agent before and after having received it. A more detailed discussion about the semantics issues is given in Section 4.1.

Common Knowledge and Ontology Definition. A CL establishes the rules used to express the content of a communication between agents, i.e., to specify the knowledge interchange format. The description of a partial world state that the agents are communicating about may contain references to *objects*, *actions* and *functions* (i.e., object-models) in one or more domains. In our example, agent *A* makes use of the FIPA-SL language in combination with the *hotels-ontology*. An ontology provides a vocabulary (class-model) for representing and communicating domain dependent knowledge, including a set of relationships and properties that are valid for the elements identified by that vocabulary. The way ontologies can be formally defined, represented and implemented is currently not fully standardised and it represents one of the most challenging aspects that many software developers have to face. This is particularly crucial for agents in *open* environments where the interoperability with other software entities is strongly dependent on the ability to share a common understanding of the specific context/domain. For this purpose, a number of different communities is currently considering the problem of providing notations and methodologies for data structures and semantics [4].

4 Experimental Feedback

Despite the existence of standard specifications for the design of agent communication components, the way to concretely implement them is not always straightforward. Full interoperability can only be achieved under additional (implicit or explicit) assumptions and off-line developers coordination. Before discussing in more details the main issues we experienced, a brief review of two specific applications enables the main communication requirements we faced to be better highlighted.

```

(request
:sender (agent-identifier: name A)
:receiver (agent-identifier: name B)
:language FIPA-SL
:ontology (hotels-ontology)
:protocol (fipa-request)
:content
((action (agent-identifier: name B)
(booking-hotel (:arrival 25-06-2002) (:departure 28-06-2002))
))
)

```

Figure 3: The message sent by agent A for requesting agent B to make a hotel reservation.

- Multi-Provider service provisioning.** The *Network Provider Interworking* (NPI) system proposes an open and distributed framework for the allocation of service demands spanning distinct communication networks. Autonomous agents acting on behalf of different network operators coordinate their actions by means of distributed constraint satisfaction (DCS) techniques integrated with economic mechanisms for automated negotiations [2]. For this purpose, agents must be able to represent in a common and standardised way the services they are offering/requesting (i.e., service model). In the NPI context, this has been achieved by: (1) defining an ad hoc CL and a specific ontology collecting terms, definitions and concepts needed for agents to understand the DCS formalism and the deployed service model; (2) deploying these ad hoc elements in combination with standard components such as FIPA compliant INPs and the FIPA-ACL for full coordination⁴. The use of ad hoc elements (which are implicitly assumed to be known by all NPI agents) enables the specific domain knowledge to be effectively captured and shared. On the other hand, the combination with standard communication components guarantees the potential interoperability with all possible external agents that (in some way) should additionally recover the needed ad hoc elements.
- Electronic financial transactions.** Considering the online business world, software agents can evaluate and optimise the utility of specific actions (such as bidding, offering, selling, etc.), gather and process huge amounts of information and follow specific strategies more efficiently and more rapidly than humans or traditional mechanisms can do. In order to better evaluate the main potential and the major limits of an agent-based e-financing framework, we developed the *Financial Agent-based Transaction* (FAT) system [9]. Here, several software entities can interact: personal assistant agents acting on behalf of final end-users, agents representing banks or financial institutions and agents providing medical insurance services. In this context, agents are assumed to make use of standard communication components (from FIPA INPs down to FIPA-SL as common CL) referring to two specific ontologies that we have defined: a *bank-ontology* modelling banking services and an *insurance-ontology* modelling medical insurances services⁵. Also in this case, the combination of ad hoc ontologies with the deployment of standard components represents the optimal trade-off (given agents resources and application requirements) for our implementation.

To summarise, both these environments are characterised by the coexistence of heterogeneous software entities that, despite self-interests, need to exhibit cooperative behaviour and autonomously interact, eventually without the direct human intervention.

4.1 Discussion

Based on our experience, the main challenges that developers have to meet for effective agent coordination can be summarised as follows:

- Agents have to be able to face different possible conversations not necessarily foreseen a priori, i.e., they should have the capability of dealing with a diversity of contexts, including heterogeneous services, agent roles and domains descriptions. For this purpose the use of standard communication interfaces, such as the ones specified by FIPA, becomes a crucial factor in all open environments increasing the potential interoperability with agents developed in other frameworks. However, the deployment of standards becomes effective only if agents and components are designed to deal with *“unanticipated requests and can generate requests for assistance if they find themselves in difficulty”* [7]. This is

⁴For more details about the implementation of the NPI system, see Appendix C of [2].

⁵For more details about the implementation of the FAT system, see <http://liawww.epfl.ch/~calisti/FAT/>

strongly dependent on how the agents decision making process is defined in relation to the possible uncertain events happening in the environment they are embedded in.

- The deployed communication components should be flexibly adaptable to different kinds of scenarios, and minimise the impact on the feasibility of specific solutions, by not imposing too hard constraints in terms of computational complexity. A layered approach (such as the one discussed in this paper) enables the decomposition of design and implementation issues by simplifying the formalisms that can be used at every layer of the CS. This has of course to be traded off against the need (complexity) of having reasoning systems able to deal with every deployed formalism.
- The context agents are embedded in, as well as their mental state, change in general as the conversations move on. This implies the need of evolving the common ontology knowledge and eventually of defining and deploying new interaction protocols. The process of dynamically modifying the adopted communication components is a very complex issue that has not been completely solved in the agent world. Some interesting ideas are currently under discussions in several frameworks such as the Agentcities.RTD project⁶.

From syntactic to semantics interoperability. For meaningful communications, distinct agents not only need to share a common representation (syntax) of all the deployed components at every level of the CS, but they also need to know how to extract the meaning of every component and thereby of the whole conversation. Therefore, in order to ensure full interoperability it is necessary to provide the semantics (i.e., the process of ascribing meaning) of all the deployed communication components.

In the agent world, to describe the mental state of an agent before sending (i.e., pre-conditions) a message and after having received it (i.e., post-conditions) is a useful way to ascribe meaning to communication primitives. The *mental state* of an agent is an intentional description making reference to beliefs, desires, intentions and other modalities that agents may have. Pre-conditions and post-conditions are usually expressed in modal logic (see Appendix A of [16]). The main challenge in this context is not only to provide a clear formalism for expressing the semantics in an unambiguous way for human developers (this is valid at all levels of the CS), but also to make sure that the agents process of interpreting the meaning of the message is compliant to the given semantics. This is an open issue that also FIPA aims to consider in the future: the idea is to provide *conformance testing* mechanisms to demonstrate that a given agent implementation is correct with respect to the formal model.

World heterogeneity: implicit versus explicit components choice. All systems (either agent-based or not) that communicate and work together must share common choices at the various levels of the adopted CS. Shared components (interaction protocols, ACLs, content languages or ontologies) can be implicitly or explicitly selected. *Implicit* components are typically represented only by conventions and procedures. This approach can work only when the various interacting entities are developed within the same framework (i.e., closed environment) or when developers preliminary decide upon common choices. *Explicit* components are (ideally) characterised by an explicit declarative representation in a well-defined knowledge representation language with a known semantics. For instance, to explicitly define an ontology it is necessary to design an underlying model of the domain in terms of objects, attributes, relations and possible actions (i.e., object-model). Then, token or symbols or terms that refer to these elements have to be assigned. Finally, encoding rules and constraints which capture important aspects of the domain model need to be fixed.

While the former approach is simpler from both a formal and an implementation point of view, it can only work for applications in which all possible interacting entities implicitly adhere to the same conventions in the same way. This requires the human designer coding common procedures and conventions, interpreting them and writing appropriate systems behaviour (i.e., the semantics providing the meaning and the context for a specific communication is assumed to be implicitly known by software entities). In the case of explicit choices, the formal semantics at all communication levels have to be encoded into a particular formalism and the whole interaction and/or each message have to be evaluated as a single expression. This latter approach is even more difficult to achieve in open systems, especially for the heterogeneity of world models, which are often very domain dependent, and for the coexistence of different semantics formalisms at the different levels of the CS (e.g., UML sequence diagrams for the interaction protocols, DAML+OIL or Ontolingua for domain description, etc.). Based upon concrete experience, we finally argue that for the implementation of an effective communication paradigm in distributed open systems, one of the major challenges that all developers have to face is to find the right trade-off between implicit versus explicit assumptions and choices.

⁶See for more details: <http://www.agentcities.org/EURTD/>

The “best” trade-off is strictly dependent on the developers (and agents) resources/knowledge and on the specific application requirements and domain constraints.

4.2 Related Work

Beside the work done within the FIPA community (including several agent oriented frameworks such as NPI, FAT and Agentcities), there are various approaches that are relevant to the discussion presented in this paper. Two of the most interesting are commented in the following.

In [11], the authors consider the problem of lookup services that facilitate the discovery of agents with specific capability descriptions (i.e., *service providers*). When considering open and heterogeneous environments, such as the Semantic Web⁷, although service requester may discover in some way the desired providers, they may not be able to communicate with them. Instead of assuming the adoption of a common stack as the one proposed in this paper, Payne et al. build agent communications on top of abstractions called *shallow-parsing templates*. Their deployment requires that agents are capable of parsing and interpreting the templates and that these templates are included within the capability descriptions providers advertise. The main drawback of this approach is that the number and the type of messages that two agents can exchange is restricted to those specified in the advertisements. Therefore, having reduced the number of components in the CS that agents have to agree upon for full interoperability can increase the flexibility and reduce the computational complexity, but can also drastically reduce the possible conversations that can take place.

Moduleco is an object oriented modular framework that has been designed for the simulation of market-like mechanisms and dynamic social phenomena in a multi-agent environment. In this context, *mediums* are used to formalise agent interactions. A *medium* is a communication component that is designed for encapsulating tailored communication services needed for distributed applications [1]. One of the main concerns is to provide re-usable mediums by pushing the application dependent features to ad hoc elements called *adapters*. This kind of approach is in a way quite similar to the CS adopted in the agent world. We therefore believe that also when deploying mediums, one of the main challenges that developers have to face for effective communication is to find the right trade-off between how much of domain dependent aspects goes into the adapters and how this specific domain knowledge can be shared by distinct software components populating the system.

5 Conclusion

In distributed systems (agents and non), the most commonly adopted approach for communication builds on top of a multi-layered infrastructure that enables decoupling of low-level data transport issues from high-level semantic interoperability aspects. This kind of structure enables the definition of communication components that have the potential to be re-used and flexibly adapted to different kind of applications and domains. Moreover, since agent oriented approaches can be considered as a more abstract paradigm for systems design and implementation that can build on top of traditional existing software frameworks, we argue that agent communication components have the potential to be re-used (and possibly modified) as middleware components for supporting communication in distributed (eventually non-agent) architectures. However, there are various open issues that need to be solved for both agent and non-agent based systems.

Acknowledgements. Many thanks to Rachid Guerraoui for useful discussions and precious suggestions. The author is also very grateful to Stefan Brantschen and Steven Willmott for their valuable comments and feedback.

References

- [1] A. Beugnard. Communication services as components for telecommunication applications. In *Workshop on Objects and Patterns in Telecom (ECOOP 2000)*, Cannes, France., 2000.
- [2] Monique Calisti. *Coordination and Negotiation for Solving Multi-Provider Service Provisioning*. PhD thesis, Swiss Federal Institute of Technology of Lausanne, 2001.

⁷For more information about Semantic Web please refer to: <http://www.w3.org/2001/sw/>

- [3] P. Th. Eugster, R. Guerraoui, and J. Sventek. Distributed asynchronous collections: Abstractions for publish/subscribe interaction. In *14th European Conference on Object Oriented Programming (ECOOP 2000)*, Cannes, France., 2000.
- [4] D. Fensel. In Dieter Fensel, editor, *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, 2000.
- [5] Tim Finin. Specification of the KQML Agent-Communication Language – plus example agent policies and architectures, 1993.
- [6] F. Giunchiglia, T. Walsh, and A. of. A theory of abstraction. *Artificial Intelligence*, 56(2-3):323-390, 1992., 1992.
- [7] N. R. Jennings. Building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.
- [8] Nicholas R. Jennings and Michael Wooldridge. Agent-Oriented Software Engineering. In Francisco J. Garijo and Magnus Boman, editors, *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, volume 1647. Springer-Verlag: Heidelberg, Germany, 1999.
- [9] Calisti M., Deluca D., and Ladd A. An agent-based framework for financial transactions. In *Proceedings of the 1st Workshop on Agent-Based Approaches to B2B, Autonomous Agents (Agents'01)*, 2001.
- [10] Pattie Maes. Artificial life meets entertainment: Lifelike autonomous agents. *Communications of the ACM*, 38(11):108–114, November 1995.
- [11] T. R. Payne, M. Paolucci, R. Singh, and Katia Sycara. Communicating agents in open multi agent systems. In *First GSFC/JPL Workshop on Radical Agent Concepts (WRAC)*, 2002.
- [12] Stuart J. Russell and Peter Norvig. *Artificial Intelligence. A Modern Approach*. Prentice-Hall, Englewood Cliffs, 1995.
- [13] M. Wooldridge. Agent-based software engineering. *IEE Proceedings Software Engineering*, 144(1):26–37, 1997.
- [14] Fipa XC00001. Fipa abstract architecture specification. *Foundation for Intelligent Physical Agents*, 2000.
- [15] Fipa XC00026. Fipa request interaction protocol specification. *Foundation for Intelligent Physical Agents*, 2000.
- [16] Fipa XC00037H. Fipa communicative act library specification. *Foundation for Intelligent Physical Agents*, 2000.
- [17] Fipa XC00061. Fipa acl message structure specification. *Foundation for Intelligent Physical Agents*, 2000.