A Formal Framework for Interaction Protocol Engineering

Fernando Alonso, Sonia Frutos, Genoveva López, and Javier Soriano

Facultad de Informática, Universidad Politécnica de Madrid,
28660 Boadilla del Monte, Madrid, Spain.
{falonso, sfrutos, glopez, jsoriano}@fi.upm.es

Abstract. This paper presents a formal framework devised to support interaction protocol (IP) engineering. The proposed framework is organized into three views that consider all the stages of a protocol engineering process, i.e. the design, specification, validation, implementation and management of IPs. The *modeling view* allows visual IP design. The *specification view* automatically outputs, from the design, the syntactic specification of the IPs in a declarative-type language called ACSL, which improves IP publication, localization and machine learning by agents. Finally, the *implementation view* provides a formal operational semantics for the ACSL language. This semantics allows protocol property verification and eases automatic code generation from the ACSL specification for the purpose of simulating code execution at design time, as well as improving and assuring correct IP compliance at run time.

1 Introduction

In the context of multi-agent systems (MASs), the approach to interaction based on *Speech Act Theory* [6] is one of the most widely used mechanisms for supporting activities such as information, task and resource sharing, action coordination and distribution, conflict resolution and agreement management. This approach assumes that agents carry out *communicative actions* in the same manner as other actions, i.e. to further their intentions, in an attempt to appropriately influence the mental state and behavior of other agents. On the basis of these premises, several *Agent Communication Languages* (ACLs) have been conceived that model communicative actions, also referred to as *speech acts*, as typed messages that constitute the building blocks for communication. KSF's KQML [7] and FIPA-ACL [8] are the two most significant examples of ACLs.

However, agents do not participate in isolated message exchanges, they enter into *conversations*, i.e. coherent message sequences designed to perform specific tasks that require coordination, such as negotiations or agreements. This exchange sequence may emerge spontaneously [12] or have been agreed upon beforehand and specified by means of an *interaction protocol* (IP) [11]. A priori specification of IPs and agreement on the particular protocol to be used in a given conversation eases the design of agents capable of entering into coherent conversations with other agents in open environments. The importance of IPs in

the design of an agent society is evident not only from their fitness for structuring behavior, but also as an organizational factor [1].

This *a priori* approach necessarily depends on the provision of a *framework* to support the modeling of interactions between agents that considers all the stages of a *protocol engineering* process, i.e. the design, specification, validation, implementation and management of IPs, considered as resources. Some aspects to be taken into account when building such a framework are:

- The ease of modeling the communicative agent behavior, mainly, the behavior of agents that obey complex interaction patterns, especially open (in terms of number of roles and/or agents) and concurrent patterns.
- Protocol maintainability and ease of reuse at both the design and specification level.
- Reliability, from the viewpoint of design validation and property verification and as regards assuring proper protocol compliance by participant agents.
- Availability and accessibility of both the protocols (i.e. designs and specifications) and ongoing conversations (i.e. protocol instances, protocol state and participant agents). This aspect is related to agent interoperability in terms of interaction.
- Scalability of both the designs and specifications (ease of composition) and the ongoing conversations for adaptation to large MAS.

This paper presents a formal framework, organized into three views, which deals with all these aspects at the IPs architectural design, formal specification and implementation level. The qualities of the developed framework have been extensively tested as part of research project *TIC2001-3451*[1], the purpose of which was to develop an architecture, based on the cooperative and strongly distributed paradigm, to manage telecommunications infrastructures holonically by means of autonomous agent societies.

The remainder of the paper is organized around the three views of the proposed framework as follows. Section 2 presents an overview of the framework. We then describe the fundamental elements of first the *modeling view* (section 3) and then the *specification view* (section 4) in detail. Section 5 deals with the *implementation view*. Finally, section 6 compares our approach with other related work.

2 Overview of the Proposed Framework

The problem of IP specification is not new to MAS developers, and a wide range of solutions have been proposed (cf. [5]). We find, however, that there is a huge void between the existing proposals based on formal techniques, whose design is extremely complex (e.g. colored Petri nets [9]), and the graphic notation-based techniques (e.g. AUML [10]), which are devoid of precise semantics and

[1] Work on this project has been partially funded by the Spanish Ministry of Science and Technology.

rule out automatic specification exchange and interpretation for the purpose of specification simulation, validation and execution. The proposed framework intends to fill this gap by means of three interrelated views:

- The *modeling view* eases the visual design of IP architecture by means of an AUML-based graphic notation. The proposed notation extends existing AUML and furnishes this notation with formal semantics. This is essential for developing the *specification view*.
- The *specification view* automatically outputs the syntactic specification of an IP from its architectural design in a declarative-type language, developed by the authors, called ACSL [2]. This improves IP publication, localization and machine learning by agents. ACSL is an abstract syntax for which an XML grammar has been developed by means of the XML Schema formalism, in order to be able to validate the specifications syntactically, and to make easier their use in Internet environments.
- The *implementation view* is based on the provision of an structural operational semantics (SOS) for the ACSL language. The developed formalism allows us to verify the properties of the designed IPs, such as their termination in finite time, conversational state reachability or the absence of deadlocks or starvations. On the other hand, the developed formalism automatically outputs code from the ACSL specification for the purpose of (1) simulating protocol execution at design time and (2) improving and assuring correct IP compliance at run time.

Figure 1 shows the products of the IP engineering process and the tools of the proposed framework. These tools allow: (1) the visual composition of IPs in $AUML^*$ notation, (2) automatic ACSL specifications generation(using an XML grammar) for models built in $AUML^*$, (3) the output of a semantic interpreter associated with these specifications, and (4) the generation, by means of code reflection techniques, of conversational proxies that improve IP compliance at run time.

3 Modeling View: $AUML^*$ Notation

To support the modeling phase, the developed framework includes a graphic notation, called $AUML*$, supported by a visual modeling tool. The proposed notation is an extension of the notation used in the AUML initiative *Protocol Diagrams*, which has also been furnished with formal semantics.

The *Protocol Diagrams* proposed by the AUML initiative are extremely helpful for designing IPs. Nevertheless, while AUML notation is fine for providing a simple representation of the designer's view of an IP, it is a basic and semiformal notation (just consider the widespread use of informal annotations). It is, therefore, inadequate as a starting point for automatically outputting the syntactic expression of the IP in a formal specification language as developed in the proposed framework.

First, we had to extend AUML notation to include other important aspects that need to be taken in account in complex interactions:

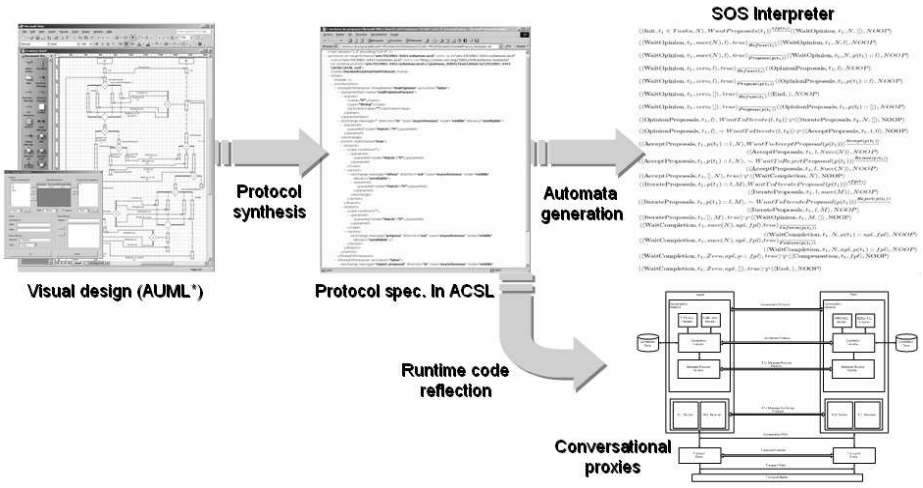

Fig. 1. Tools and products of the proposed framework

Correlation and causality Generally, agents participate in more than one dialogue at the same time. Therefore, relationships of correlation and causality between messages need to be specified so that each message can be associated with the specific IP instance to which it belongs, and messages from different IPs involved in the interaction can be related.

The time factor The time factor needs to be considered in IP design. It should be possible to specify a pause between exchanges for a finite time and establish deadlines for both an exchange sequence and for the global interaction.

Protocol exceptions Apart from describing the basic message exchange pattern, a protocol diagram should also consider the interaction flow triggered by the occurrence of a protocol exception as a result of out-of-sequence message reception, message loss, time-outs, etc.

Event management An IP should consider the occurrence of given events as triggers of a given exchange pattern.

Compensation protocols It should also be possible to express *transactional contexts* within the main exchange pattern and associate compensation protocols (cancellations, renegotiations, etc.) with such contexts.

Figure 2 shows a screenshot of the $AUML^*2ACSL$ tool which has been developed as part of the proposed framework. It illustrates the Iterated Contract Net protocol design, proposed by FIPA [11], in $AUML^*$ notation. The *symbol palette* shows the new constructs. Section 4 explains what they are used for in more detail.

Apart from the AUML extension, this notation had to be formalized semantically as a provision for formalizing the other views of the proposed framework, principally the *specification view*. To be able to output an ACSL specification, we needed a lot of information not covered in the notation proposed by AUML

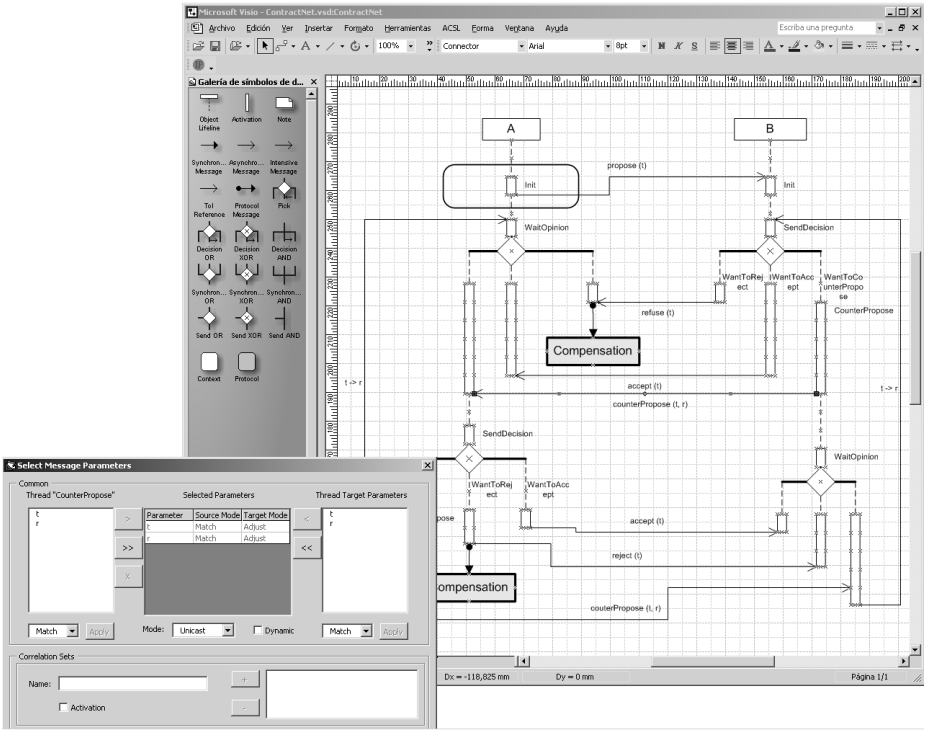

Fig. 2. $AUML^*2ACSL$ tool screenshot

or, alternatively, expressed informally by annotations, apart from additional constructs not included in AUML notation. Figure 2 also shows a dialogue including some semantic elements associated with an ACL *counterPropose* exchange. The meaning of these elements is described in section 4.

4 Specification view: ACSL language

ACSL defines an abstract syntax that establishes a vocabulary that provides a standard formal description of the contractual aspects of IPs modeled using $AUML^*$ for use by design, implementation and execution monitoring libraries and tools. ACSL separates internal agent IP implementation from its external description. This is a key point for improving communication interoperability between heterogeneous agent groups and/or agents that run in heterogeneous agencies (platforms). It is based on ACL messages specifying the message flow that represents an IP between two or more agents and requires no special-purpose implementation mechanism. The following sections describe briefly the structure of ACSL. See [2] for a more detailed description of the language.

Figure 3 illustrates the overall structure of a protocol specification in ACSL.

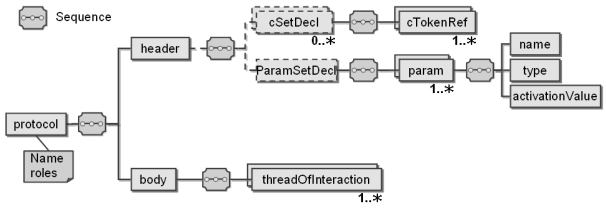

Fig. 3. Syntactic organization of a *protocol* specification

It shows how the specification is composed of a *name*, a *header* and a *body*, all defined in the context of a block element *protocol*. The *name* element identifies the protocol for the purpose of referencing from other specifications in which it is to be embedded or with which it is to be interlinked. The *header* element declares the correlation sets (*cSetDecl*) and the properties (*paramSetDecl*) used in the message exchanges for correlation and dynamic linking and to specify the semantic elements, respectively. The body of the protocol contains the specification of the basic exchange pattern(*body* block). This item is formed by the composition of many *threadOfInteraction* elements that fork and regroup to describe the communicative behavior of the agent. The *threadOfInteraction* element is used to directly specify an exchange pattern or reference a protocol definition included in another specification by means of a qualified name. A *threadOfInteraction* (Figure 4) combines zero or more basic actions, references to subprotocols, conditional and iterative constructs and other *threadOfInteraction* that are interpreted sequentially, in the same order in which they are referenced.

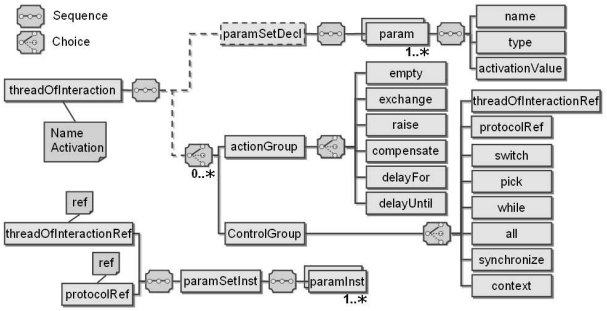

Fig. 4. Language constructs for specifying a *thread of interaction*

Message exchanges (*exchange* element) are the fundamental atomic actions in agent interaction. ACSL includes only the exchange properties that are part of the protocol specification.

A *threadOfInteraction* eases the composition of an exchange pattern by means of a set of control constructs (*ControlGroup* in figure 4) that express conditional, concurrent and iterative interaction flows.

Switch: A *switch* expresses a conditional behavior equivalent to the XOR connector in AUML notation. The Boolean *multiChoice* attribute allows the *switch* to also express conditional behavior equivalent to the OR connector in AUML notation.

While: The *while* construct repeats the exchange pattern determined by a thread of Interaction an undefined number of times. The specified behavior is executed until the given (*condition*) is no longer true. The condition is opaque, as discussed earlier.

All: The *all* construct expresses the concurrent execution of a set of interaction flows that is not subject to any time order. *All* expresses the semantics of the AND connector in AUML notation.

Pick: The *pick* construct (figure 5) expresses precondition waits. It waits for the reception of an event (or set of events) and then executes an exchange pattern associated with this event. The possible events are message reception and end of a *delay* action. If the *times* modifier is specified, *pick* will be repeated a predefined number of times (for example, to specify wait for n messages) before finally executing the exchange pattern associated with the construction *onTimes*.

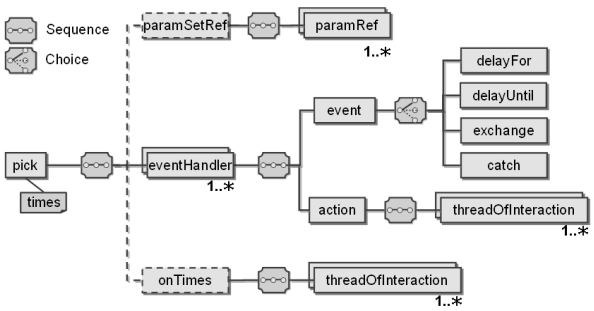

Fig. 5. Language constructs for the *pick* element

Repeat: The *repeat* construct repeats the exchange pattern given by a thread of interaction a pre-established number of times. The actual number of times it is repeated is opaque, i.e. is not part of the ACSL specification.

Synchronize: The *Synchronize* construct establishes the set of threads of interaction that should be synchronized after an *All*, a multiple-choice *Switch* or an*Or*.

The atomic action *raise* in Figure 4 represents the firing of an exception related to the IP and can model real situations that call for cancellation or abnormal or unexpected termination of the protocol in question. The *raise* action

causes the invocation of the IP that allows the agents to recover from abnormal situations or finish if recovery is not possible. This protocol is specified by means of an *exception* block element declared after defining the flow pattern (*threadOfInteraction* block) of the *context* with which the exception is to be associated.

In the event of an error occurring during a complex interaction that involves one or more protocols, the agent involved should be given the chance to start up a conversation characterized by one or more appropriate IPs that *compensate*, as far as possible, for this error, taking the system back to a state similar to what it was at the start of the interaction. This conversation is called *compensation protocol*. Figure 4 showed the language constructs that participate in the description of this behavior. A compensation protocol is defined in a context by means of a named *compensation* block. A *compensation* block is explicitly invoked by means of the *compensate* process that references the block name. Accordingly, the initiation of the compensation protocol by this agent can be asynchronous with respect to its main exchange pattern.

5 Implementation view: Specifying the dynamics of IPs

The definition of an XML grammar for ACSL by means of the *XML Schema* formalism can only validate the IP specifications syntactically. To be able to validate and evaluate these specifications semantically, the ACSL language needs to be furnished with formal and operational semantics that can unambiguously describe the dynamic meaning of its syntactic constructs.

The features of ACSL have led to the use of the concept of *Structural Operational Semantics* (SOS) [4] as an approach for specifying the dynamic meaning of IPs. The *dynamic meaning* of a protocol is obtained from the dynamic meaning of the different syntactic constructs that appear in its specification. It covers the execution of the specification, including expression evaluation, message sending and reception and the execution of other non-communicative actions.

The SOS denotes a formalism that can specify the meaning of a language by means of syntactic transformations of the programs or specifications written in this language. Some special points had to be taken into account to apply the SOS formalism, designed for programming languages, to a specification language such as ACSL. The definition of operational semantics suited for ACSL represents a three-step process:

1. Definition of a terminal and labeled transitional term-rewriting system based on the operational semantics described in [3],
2. Definition of the interpreter I for this system, as proposed in [4], whose behavior is specified by a set of production rules. I is modeled as a function whose argument is a protocol P specified in ACSL in an environment ω, and describes the behavior of $\langle P, \omega \rangle$ as an (in)finite series of productions like $\langle P, \omega \rangle \rightarrow \langle P_1, \omega_1 \rangle \rightarrow \langle P_2, \omega_2 \rangle \rightarrow \ldots$ If P ends, then the result is $\langle END, \omega_n \rangle$.
3. Process of outputting the interpreter for each ACSL construct.

The provision of formal semantics for ACSL means that the IP specification can be analyzed to find out whether the IP has certain properties, such as termination in finite time, conversational state reachability or no deadlocks and starvations. On the other hand, the provision of operational semantics makes it possible to automatically derive IP implementation from protocol specification, easing its simulation and the automatic generation of proxies that assure that each participant effectively complies with the protocol rules and provides assistance for protocol machine learning. The details of this view are left for a later paper.

6 Related Work and Conclusions

Other work has been completed along the lines presented in this paper. Thus, [13] presents a coordination language called COOL, which is founded on speech act-based communication and a structured conversational framework that captures the coordination mechanisms used by agents. COOL has been used to design and validate coordination protocols for multiagent systems. The coordination activity is modeled as a conversation between two or more agents and is specified by means of a declarative language whose syntax is intrinsically linked to the finite state machine model, which is an impediment to its use in concurrent coordination environments. More recently, [14] explored the conversion of COOL policies to Petri Nets to enable their analysis. The language provides support for conversation embedding, error recovery and conversation continuation, but it does not cover aspects like message correlation and obscures specification. [15] presents the Z specification language, which is applied to contract-net protocol formalization as a goal-directed system. The *Agentis* framework presented later in [16] also specifies the IPs in Z. [17] presents the AgenTalk coordination language, which adds the notion of inheritance to conversation specification. [18] introduces the notion of a conversation suite as a collection of widely used conversations known by many agents and [19] uses definite clause grammars to specify conversations. Finally, [1] proposes a coordination model for specifying IPs, possibly open and concurrent. The model is supported by a high-level Petri Net-based Object-Oriented language called CoOperative Objects (COO) that covers all the steps of protocol engineering, enabling it to express concurrency or validate and simulate the specified protocol. But this is at the cost of obscuring their interpretation, because the model does not have a graphic representation close to the designer's view of the protocol. Although each of these approaches contributes in one way or another to our general understanding of the concept of framework, more work is needed to improve the design, sharing and use of IPs by agents. This paper has attempted to contribute to this effort by presenting a formal framework organized as three viewpoints which improves IP modeling, publication, localization, exchange and machine learning by agents. It also allows protocol property verification and eases automatic code generation (protocol synthesis) for the purpose of simulating code execution at design time, as well as improving and assuring correct IP compliance at run time.

References

1. Hanachi, C., Sibertin-blanc, C.: Protocol Moderators as Active Middle-Agents in Multi-Agent Systems. In Autonomous Agents and Multi-Agent Systems, Vol. 8. Kluwer Academic Publishers, The Netherlands (2004) 131-164
2. Soriano, J., Alonso, F. and López, G.: A Formal Specification Language for Agent Conversations. In: Marik, V., Mller, J.P. and Pechoucek, M. (eds.): Multi-Agent Systems and Applications III. LNCS 2691, Springer-Verlag, Berlin (2003) 214-225
3. Eijk, R., Boer, F., Hoek, W. and Meyer, J-Ch.: Operational Semantics for Agent Communication Languages. In Dignum, F and Greaves, M. (eds.): Issues in Agent Communication, LNCS 1916, Springer-Verlag, Heidelberg (2000) 80–95
4. Plotkin, G.D.: A structural approach to operational semantics. Technical Report DAIMI FN-19. Aarhus University, Computer Science Department, Denmark (1981)
5. Dignum, F., Greaves, M. (eds.): Issues in Agent Communication. LNCS 1916, Springer-Verlag, Heidelberg (2000)
6. Searle, J.R., Vanderveken, D.: Foundations of Illocutionary Logic. Cambridge University Press, Cambridge (1985)
7. Finin, T. et al. Specification of the KQML agent-communication language. ARPA Knowledge Sharing Initiative, External Interfaces Working Group (1993)
8. FIPA-Foundation for Intelligent Physical Agents: FIPA ACL Message Structure Specification. http://www.fipa.org/specs/fipa00061, FIPA (2002)
9. Cost, R.S. et al. Modeling agent conversations with CPN. In Bradshaw, J. (ed.): Workshop on Specifying and Implementing Conversation Policies (1999) 59–66
10. Odell J. et al. Representing agent interaction protocols in UML. In Ciancarini, P. and Wooldridge, M. (eds.): Agent Oriented Software Engineering (AOSE'00) Workshop, Limerick, Ireland (2000) 121–140
11. FIPA-Foundation for Intelligent Physical Agents. FIPA Interaction protocol Library Specification. http://www.fipa.org/specs/fipa00025, FIPA (2003)
12. Dignum, F.: From Messages to Protocols. In Dignum, F. and Sierra, C. (eds.): European Perspective on Agent Mediated Electronic Commerce. Springer-Verlag, Heidelberg (2000)
13. Barbuceanu, M and Fox M.S.: COOL: A Language for Describing Coordination in Multiagent Systems. In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95). AAAI Press, San Francisco, USA (1995) 17–24
14. Galan, A. and Baker, A.: Multi-agent communications in JAFMAS. Workshop on Specifying and Implementing Conversation Policies, Washington (1999)
15. d'Inverno, M. and Luck, M.: Formalising the Contract Net as a Goal-Directed System. In W. V. de Velde and J. Perram (eds.): Agent Breaking Away. LNAI 1038. Springer-Verlag, Heidelberg (1996) 72–85
16. d'Inverno, M., Kinny, D. and Luck, M.: Interaction Protocols in Agentis. In Proceedings of the International Conference on Multi-Agent Systems ICMAS'98 (1998)
17. Kuwabara, K. et al.: AgenTalk: Describing multiagent coordination protocols with inheritance. In Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence (ICTAI'95), Herndon, Virginia (1995)
18. Bradshaw, J.M. et al.: Kaos: Toward an industrial-strength open agent architecture. In Bradshaw, J.M. (ed.): Software Agents. AAAI/MIT Press, Cambridge (1997) 375–418
19. Labrou, Y. and Finin, T.: Semantics and conversations for an agent communication language. In Huhns, M. et al. (eds.): Readings in Agents. Morgan Kaufmann (1997) 584–591