Integrating Jason in a Multi-Agent Platform with Support for Interaction Protocols

Bexy Alfonso Emilio Vivancos Vicent Botti Ana García-Fornes

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Valencia (Spain)
{balfonso,vivancos,vbotti,agarcia}@dsic.upv.es

Abstract

Agent communication is a core issue when studying all possible ways for agents to organize and collaborate to achieve their goals. We can count on communication standards, as the FIPA Interaction Protocols. On the other hand we can count on high level agent programming languages, like AgentSpeak, which allow us to model and represent the agent and its knowledge and behavior. In this paper we present a proposal to add to Jason (an interpreter of an extended version of AgentSpeak) a new level of abstraction in the task of programming conversations between agents. The agent communication follows the FIPA interaction protocols. A new entity called Communicator Manager acts as an interface between the agent programming language (Jason) and the platform communication facilities (Magentix 2). This approach allows the programmer to focus on programming the agent knowledge and reasoning parts instead of the interaction protocols. An agent can call the communication manager to start a conversation. The communication manager will control the different steps of the conversation and will modify the agent belief base to represent the results of the different steps of the conversation protocol. Therefore, the agents can use this knowledge in its reasoning process. This approach can be easily transfered to others agent programming languages and platforms.

Categories and Subject Descriptors I [2]: 5

General Terms Languages

Keywords Interaction protocols, agents, agents conversations, FIPA, Jason, agents communication

SPLASH'11 Workshops, October 23–24, 2011, Portland, Oregon, USA.

1. Introduction

Nowadays multi-agent systems is one of the most active and promising research fields in Computer Science. Multi-agent systems provide adaptability, scalability, distribution, fault tolerance, intelligence, and autonomy. All these features make multi-agent technology an interesting approach for a wide set of applications. Currently research topics on multi-agent systems include definitions, standards, methodologies, programming languages, semantics, platforms and communication. One interesting area in the development of multi-agent systems is the study of agent programming languages [5, 9, 14] and more specifically all the language aspects related with the agents communication. An agent in a multi-agent environment is a software entity which needs to communicate with other agents in order to achieve its goals. This communication is a crucial part when developing agents. An agent developer usually spends a big amount of time programming the part of the agent that allows the agent to maintain a conversation with other agents.

The two main agent communication languages proposals are the USA DARPAS's Knowledge Query and Manipulation Language (KQML) [8] and the FIPA ACL [11]. In this work we focus in the second language because has a high degree of acceptance in the agent programming community.

A basic FIPA-ACL message is composed of several parameters including a performative (indicating the type of the communication act of the message), the sender, the receiver, the content, the used ontology, . . . A conversation between agents is defined as a sequence of messages exchanged by them. Obviously one of the most complex task programming an agent conversation is to study all the possible sequences of messages exchanges that can be performed during the conversation. The Foundation for Intelligent Physical Agents(FIPA) has defined a set of standard conversations containing predefined patterns of valid message sequences. These patterns have been defined as general as possible to allow to apply them in different situations and environments. These protocols are called *FIPA Interaction Protocols* [12]. The FIPA has defined a set of basic predefined standard in-

teraction protocols including the more typical conversations followed by the agents, but other ad hoc protocols could also be defined. The set of FIPA interaction protocols includes: ”request”, ”query”, ”request when”, ”contract net”, ”iterated contract net”, ”English auction”, ”Dutch auction”, ”brokering”, ”recruiting”, ”subscribe” and ”propose”.

In this paper we present how to integrate the FIPA interaction protocols in Jason taking advantage of the platform facilities. Jason [6] is an interpreter for an extension of the AgentSpeak programming language [17]. It is written in Java and its creators are R. Bordini and J. Hubner. An agent in Jason is programmed using this extension of the AgentSpeak language but it is also possible to write some parts of the agent in Java. For instance, the programmer can write some code in Java to program the environment, or to construct Jason “internal actions”. Those internal actions are used to extend the language and they are designed to be executed as “being outside the agent’s mind”. This execution is done as one step of the agent reasoning cycle. Jason also provides agent communication based on speech-acts. Unfortunately it is the programmer’s responsibility to write the communicative actions to implement all steps included in the conversation. The main objective of our proposal is to facilitate as much as possible the creation of agents that communicate with others using the FIPA standard interaction protocols.

The rest of the paper is organized as follows: In the next section a set of related works is presented. Section 3 introduces the guidelines followed to make the design of the proposal and the design description itself. In section 4 the main components that are part of the implementation are shown. Finally section 5 presents a discussion and future work.

2. Related works

Jade [1, 4] is a FIPA compliant platform that allows to communicate agents using the FIPA standards. That means that messages in Jade follow the format specified by FIPA. Moreover, Jade provides Java classes to handle all the FIPA interaction protocols. The *jade.proto* package provides some Java classes containing several callback methods that can be used by programmers. These methods can be redefined by adding the required logic to solve a specific problem. The classes in the *jade.proto* package are divided into two groups depending on the role performed by the agent using the class: initiator or responder. The agent interactions must be programmed also in Java by using the constructions provided by the platform. This leaves out the possibility of building behaviors for those interactions in a language with a higher level of abstraction.

Jadex [7, 16] is a multi-agent platform following a typical BDI model. Originally Jadex was implemented to be executed under Jade, but currently it can be executed alone or under other communication platforms using *adapters*. A Jadex agent is composed of an agent definition file written in XML and the Java classes that implement it. Jadex agents could be FIPA compliant due to JADE, and both JADE and Jadex agents can survive in the same platform. One of the tools contained in Jadex is the “Interaction Protocols Capability” with offers built-in support for most of the FIPA interaction protocols. In Jadex, the different steps of the protocols have been analyzed and led to the extended specification of goal-oriented protocols [2]. A protocol is started by creating an instance of an initiator goal (“rp_initiate”), setting the needed parameters, and dispatching it. There are also three goals for the participants: “rp_receiver_interaction”, “rp_decide_request” and “rp_execute_request”). When the protocol has ended, the results are returned as out-parameters of the goal.

Jadex and Jade provide Java classes for implementing FIPA interaction protocols, so the programmer can’t use other specialized programming languages (like AgentSpeak) with more facilities to model and describe agents. On the other hand, none of the analyzed platforms allow to dynamically modify the steps sequence in the interaction protocol in order to create more open and flexible conversations.

Our proposal is more flexible because allows to use protocols that can be changed dynamically thanks to the facilities offered by the platform: New states and transitions between the conversation steps can be created at execution time. Other advantage of our proposal is that, as it is explained in the next section, a conversation manager stores and automatically adds many information required in the creation of messages during the conversation (initiator, participants, the state of the conversation,. . .), so the programmer does not have to include it.

3. Design

3.1 Guidelines

Due to the current necessities of using interaction protocols, the first goal of this proposal is to facilitate the task of programming this part of the agent. The objective is to have agents almost no aware that their reasoning is part of a conversation they are having; agents who don’t need to know which was the previous or the next step at any moment of the interaction protocol. Given this level of abstraction, the agent programmer just needs to focus on the reasoning, instead of using the platform specific communication primitives or the protocol specification details for managing it.

On the other hand, given the so generalized FIPA Interaction Protocols Specification, it is aimed to integrate them with a high-level language to reach this objective. The aim is to make easier for the programmers to use this standards.

Finally this proposal can be extrapolate to other agent programming languages or platforms following the same idea. This avoids the programmers to have to learn new constructors to use the communication interaction protocols and makes the proposal versatile enough to facilitate this task regardless of the tools, platform and language employed.

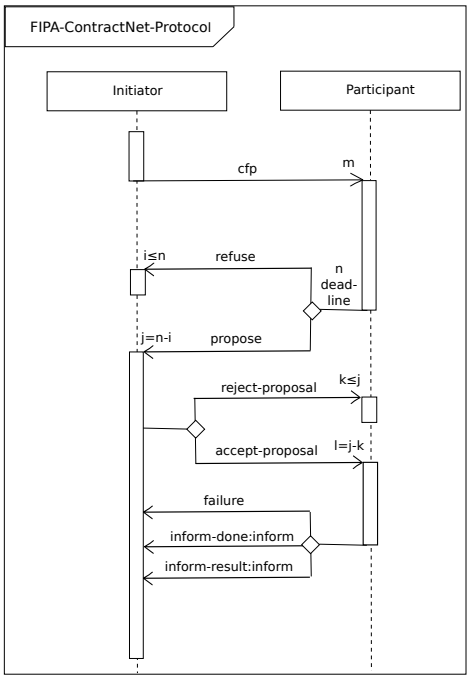

Figure 1. FIPA Contract Net Interaction Protocol. [12]

3.2 Description

Based on FIPA specifications [12] an interaction protocol can be seen as a sequence of messages between two roles involved: *initiator* and *participant*. The *initiator* begins the interaction notifying the participant with a message about it's interests, and the *participant* replies to the *initiator* requests according to the protocol characteristics. Both roles exchange messages until the end of the interaction. As example, Figure 1 shows the Fipa Contract Net Protocols specification based on a extension of UML [15]. From now on, this protocol is going to be taken as example for the further descriptions.

According to [10], a conversation can be seen as a direct graph composed of nodes and arcs; nodes represent the states or steps of the conversation and arcs represent the transitions between states. Having this, the purpose is to integrate the steps of the communication with the reasoning cycle of a BDI agent. First of all, it is necessary to know which are those steps. Figure 2 shows an example of the FIPA Contract Net Protocol steps, from the *initiator* perspective, according to [12] seen as a directed graph. In Figure 2 there are also some steps (nodes) colored; they represent the steps in which the agent can perform its reasoning and/or take decisions. The next thing to do for each role on each communication protocol is to identify where a decision must be taken or a particular reasoning must be done.

On each step a *Conversation Manager (CM)* indicates to the agent that some reasoning is necessary. The conversation is paused until the corresponding result is provided by an

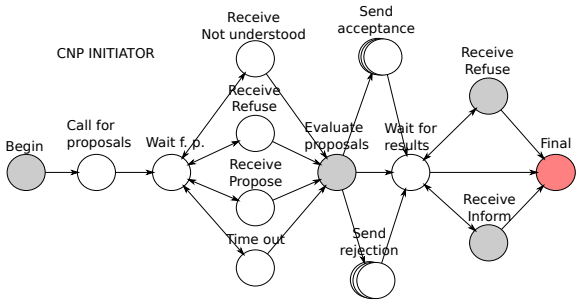

Figure 2. Steps of the Fipa Contract Net Protocol for *initiator* agent.

interaction between the agent and the CM or until a specific timeout. The reasoning cycle of the agent goes on with its normal execution independent of the conversation.

The information provided to the agent at the different steps during the communication, will act as triggering event at the same time that it provides the necessary information for the agent to reason about it. On those steps where no reasoning is necessary, the CM will follow the sequence of steps indicated in the protocol specification.

In the final step of the communication protocol the CM ends the conversation, and performs other required tasks (depending on the platform characteristics). Besides, the agent is being informed about the results. On the other hand, the agent can end the conversation at any time by interacting with the CM. If something related to the conversations fails (like exceeding the response times or if some communication error arise), the CM informs the agent.

4. Implementation

There are three main kind of technologies necessary to implement this proposal. In the first place it is necessary a platform to manage the multi-agent system, to deal with all subsequent problems arising from the interaction and to control agents performance. Secondly it is required a language for defining the agents itself and its components. In this sense, to use a language that supports the BDI agents model is an adequate choice due to the fitness of this model to agents with high reasoning skills. Finally, a CM is also necessary in order to control the interaction between the reasoning of the agent and the conversation flow.

4.1 Communication platform

There are several platforms and tools that help to build communicative agents. For our approach we have looked for a FIPA-compliant platform capable of supporting technologies and techniques related to communications between agents, and able to allow programing reasoning agents using a high level language. In this direction Magentix 2 [3] has appeared as a promising platform. This platform supports agent societies communication, security .... Besides, the Magentix 2 platform is able to deal with the distributed and

autonomous nature of agents by using technologies able to cope with the resulting dynamism and flexible interactions of this kind of systems. Among this technologies we find the support to indirect communication and interaction protocols between agents organizations. In this way Magentix 2 provides support at the three levels stated in [13]: organization level, interactions level and agent level.

4.2 Programing language

Magentix 2 provides support to program agents using a high level language, which is Jason [6]. According to the AgentSpeak model of agent, each Jason agent can be defined by three main elements: beliefs, goals and plans. Beliefs represent the agent actual vision of the world in which it is situated. They change continuously during the agent execution and there are different ways to make this happen. For example, when the agent "perceives" its environment (the world observable by it), when some information is sent to him through a message or when it explicitly modifies those beliefs as a consequence of some previous reasoning. Moreover, a Jason agent having a goal means mainly that it wants to reach certain situation in which it believes this goal is truth ("achievement goal"). Although there are other kind of goals that are constructions to retrieve information from the agent belief base("test goals"). Finally the plans in Jason allow, through a necessary sequence of steps, to reach some goal, as it can be thought intuitively. This happens in such way that the adding of this goal acts as a triggering event for this sequence of actions. If the actions doesn't fail the goal would be reached. Besides the adding of achievement goals, there are other triggering events for plans in Jason: deletion of achievement goals, adding and deletion of beliefs and adding or deletion of test goals.

The Jason Agent owns a reasoning cycle that determines what to do and how to do it in each moment. This cycle allows to create intentions based on the plans the agent has to try to fulfill, making all the necessary actions. On each reasoning cycle basically the agent must interact with it's environment, must check all messages that has been sent to him, check if there is some applicable plan according to the events that have happened and intend to fulfill them.

Another important facility provided by Jason are the internal actions. They allow to perform some processing that the agent needs to do withing it's code by accessing to legacy code (Java in this case). Internal actions are going to play an important role for the agent to interact with the Magentix 2 platform.

4.3 Conversation Manager

The support to FIPA interaction protocols by the Magentix 2 platform is given mainly by the "Conversations Factories" [10]. It's main goal is to offer the necessary tools to develop conversations based on interaction protocols, allowing to change them at execution time. The *Conversation Manager (CM)* is an interface between the platform (Magentix 2 and its *Conversation Factories*) and an agent programming language (Jason) which provides an easy way of implementing the FIPA Interaction Protocols from a higher level of abstraction.

The Conversations Factory provides two kinds of structures: *CProcessor* and *CFactory*. The first one performs the actions and manages the sent and received messages in each step of the conversation that is having an agent (even *initiator* or *participant*); this includes to determine the next step to go in each moment when the conversation is taking place. The CFactories, on the other hand, are in charge of starting the conversation and creating CProcessors corresponding to a specific protocol. The way it works change depending on the role of the agent. If the role is *initiator* the conversations can start without needing an external event. If the role is *participant*, an event is required to become part of the conversation.

As it was stated in Section 3.2, when a reasoning is necessary, the conversations is paused, waiting for an event to go on (results provided by the agent or a timeout). In this sense it is important to take into account that there are two threads interacting: the agent and the conversation. This functionality has been added by using semaphores on each conversation for each role, allowing to interact with agent beliefs base and also facilitating the agent to interact with the platform through internal actions.

4.4 Example

Returning to the example of the FIPA Contract Net Protocol, lets analyze how to implement in Jason a step were a reasoning is necessary for the *initiator* role. The steps (circles) colored in Figure 2 are the steps in which the agent must decide an action. In the contract-net protocol those steps are: *Begin*, *Evaluate Proposals*, *Receive Refuse*, *Receive Inform* and *Final*. The *Begin* step must be explicitly implemented in the agent that plays the *initiator* role so the agent must decide when to start the conversation and with which agent it is interested to interact. In the *Evaluate Proposals* step the agent must know which are the received proposals to evaluate them and chose some based on its own criterion. Finally the agent must offer those results to the CM. In Figure 3 it is shown the appearance of the code that implements this actions.

The code of Figure 3 shows the plan (written in AgentSpeak/Jason) that implements the reasoning corresponding to the *Evaluate Proposals* step. Line 1 is the triggering event of the plan, which is in this case, the adding of a belief to the agent beliefs base. This belief was "perceived" by the agent because the CM added it to the agent perceptions previously; the CM also recovers the proposals made by participants and takes care of all the necessary actions to get them. This proposals are also added previously on the agent belief base so when it is time to evaluate them, the agent knows already all he needs to know (lines 1 and 2 of Figure 4 are an example of the perceptions that the CM adds to the agent belief

```
1 +proposalsEvaluationTime(ConvID):
2     proposal(Prop,Sender,ConvID)
3 <- .print("PROPOSALS EVALUATION TIME.");
4    (...)
5    !evaluateProposals(ConvID,Acc,Rej);
6    (...)
7    .int_act_FCN_Init("PropEvaluated",Acc,
       Rej,ConvID).
```

Figure 3. Appearance of the plan for evaluating proposals written in Jason in a Contract Net Protocol for the *initiator* role.

```
1 proposal(23.0,bob,fcn).
2 proposal(13.0,amy,fcn).
3 (...)
```

Figure 4. Appearance of the *initiator* belief base before evaluating proposals.

base). In Figure 3 the term *ConvID* will contain some kind of identifier provided by the *initiator* to the CM when the conversation is started. Line 2 is the condition for executing this plan: the matching between this literal and something existing on the belief base, or, what is the same, there must be at least one proposal to evaluate proposals. Line 3 is the code to display a text on the screen; line 5 indicates the addition of a goal for evaluating the proposals, where terms *Acc* and *Rej* will take the acceptances and rejections as result of this evaluation. Finally, in line 7, an internal action is called with the result as argument and also the identifier used by the agent to know from which conversations are those results. Additionally, an indicator ("PropEvaluated") is also passed as argument, to inform about the step of the conversation in which the reasoning is.

On the steps *Receive Refuse* and *Receive Inform* of the protocol maybe it is not necessary a reasoning but the agent could perform some action depending on what happened in the conversation, whether the *participant* refuses or informs after the *initiator* sends its acceptance. The way of programming those steps are the same than the one described previously. The *Final* step it similar, but it is just colored different in Figure 2 because it is optional to "capture" the triggering event associated with the addition of the belief of ending the conversation.

4.5 Summary of the steps to use a protocol

To implement an interaction protocol in an agent using our CM, the programmer should perform the following steps:

1. Ensure that the appropriate parameters for starting the conversation are known by the agent. For example in the case of the *initiator* of a Contract Net Protocol the time out for the steps execution and the name of the participant or participants it wants to interact with must be specified.
2. Implement a plan which contains the first call to the internal action corresponding to the desired protocol (in the case of the *initiator* role). If it is the case of the *participant*, it must have a plan to join the conversation when it receives this request from the *initiator*.
3. Implement a plan for each mandatory reasoning step; this implies to have the predefined percept for a step as the triggering event for the corresponding plan. The final action of each one of this plans must be always a call to the internal action of the protocol with the appropriate parameters, which allows to execute the next step of the protocol.
4. Optionally, implement a plan for each optional reasoning step; generally this step is the final. If some kind of conversation identifiers management is done, or some post conversation actions are necessary, this is the place where it must be performed.

5. Discussion and future works

The specification of standards for interaction protocols by FIPA has promoted the development of communication mechanisms under such standards. But when facing this task, it is known that it is necessary to deal with several arising problems, with no easy solutions. When implementing interactions protocols, it is necessary to deal with synchronization issues, communications and interaction fails, information consistency and some other. This may lead, in some way, to leave aside a core issue: the information that agents must exchange and the information they must have at the end of the interaction. Our approach tries to focus in this line taking advantage of the benefits of using a high level language to program agents and leaving away all "non core" details when using interaction protocols.

This idea, in the way it is formulated, is applicable to other platforms and high level agent programming languages, so it is itself a way to promoting the use of interaction protocols under certain standards.

Specifically the conversations factory that Magentix 2 platform provides, allows also to dynamically modify the "standard structure" of the interaction protocol [10] , so by using this technologies, it is possible to create dynamic agents interactions too.

In our approach the communication is asynchronous so the sender is not blocked until the receiver agent process the received message. Therefore, the agents can maintain several conversations with different agents and/or protocols at the same time. The communication between agents is made using the FIPA interaction protocols which are directly supported by the agent programming language (AgentSpeak/-Jason) so the programmer only have to specify the agent's

knowledge needed to take decisions. The agents involved in a conversation directly receive the results of the different steps of the conversation in the form of beliefs in their belief base. Therefore, the agents can use this knowledge in its reasoning process.

It is aimed to reach modularity in such a way that the conversation structure and details not related with the information to be exchanged by the agents are not included in the own agent logic. Either implementing the whole conversation through Jason plans and communicative acts or doing it by invoking callback methods as in Jade, it is necessary to deal with details as the messages creation and preparation. On the other hand, in Jason plans it would be necessary to consider exceptions and timeouts etc, and if using Jade for example, it wouldn't be possible to exploit the advantages of a high level language made for agents with under a BDI model. With this proposal it is searched efficiency, improvement for the applications validation, reduction of the computational load by having a CM in charge of the management and execution of each conversation independently. It is also intended to reach simplicity by offering a set of already implemented protocol templates which will allow to use protocols in an easier way.

The future direction of this work is focused on elaborate a guide or methodology for extending this approach to other platforms and agent programming languages, so that the use of interaction protocols becomes a simple task, allowing to implement more sophisticated interactions in real world. It is intended too, to develop a set of protocols templates to be used by all Jason programmers over the Magentix 2 platform. Other future works include the performance evaluation and scalability of the implemented interaction protocols, by comparing the results with Jason native implementations.

Acknowledgments

This work has been partially funded by TIN2008-04446, TIN2009-13839-C03-01 and PROMETEO 2008/051.

References

[1] Jade. `http://jade.tilab.com` .

[2] Jadex user guide. `http://jadex.informatik.uni-hamburg.de/docs/jadex-0.96x/userguide` .

[3] J. M. Alberola, J. M. Such, A. Espinosa, V. Botti, and A. García-Fornes. Magentix: a Multiagent Platform Integrated in Linux. In *EUMAS*, pages 1–10, 2008.

[4] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley and Sons, 2007.

[5] R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, editors. *Multi-Agent Programming. Languajes, Platforms and Applications*. Springer, 2005.

[6] R. H. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-agent Systems in Agent Speak Usign Jason*. John Wiley & Sons, 2007. ISBN 978-0-470-02900-8. URL `http://jason.sf.net/jBook/jBookWebSite/Home.php`.

[7] L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A BDI agent system combining middleware and reasoning. In M. C. M. K. R. Unland, editor, *Software Agent-Based Applications, Platforms and Development Kits*, pages 143–168. Birkhäuser-Verlag, Sept. 2005.

[8] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *CIKM '94 Proceedings of the third international conference on Information and knowledge management*.

[9] M. Fisher, R. H. Bordini, B. Hirsch, and P. Torroni. Computational logics and agents: A roadmap of current technologies and future trends. *Computational Intelligence*, 23, 2007.

[10] R. L. Fogués, J. M. Alberola, J. M. Such, A. Espinosa, and A. García-Fornes. Towards Dynamic Agent Interaction Support in Open Multiagent Systems. In *Proceedings of the 13th International Conference of the Catalan Association for Artificial Intelligence*, volume 220, pages 89–98. IOS Press, 2010.

[11] Foundation for Intelligent Physical Agents. *FIPA XC00061D: FIPA ACL Message Structure Specification* .

[12] Foundation for Intelligent Physical Agents. *FIPA XC00025E: FIPA Interaction Protocol Library Specification* .

[13] M. Luck and AgentLink. *Agent technology : computing as interaction: A roadmap for agent-based computing. Compiled, written and edited by Michael Luck et al.* AgentLink], [Southampton, U.K. , 2005. ISBN 0854328459.

[14] V. Mascardi, D. Demergasso, and D. Ancona. Languages for programming BDI-style agents: an overview. In F. Corradini, F. D. Paoli, E. Merelli, and A. Omicini, editors, *Proceedings of WOA 2005: Dagli Oggetti agli Agenti. 6th AI*IA/TABOO Joint Workshop "From Objects to Agents"*, pages 9–15. Pitagora Editrice Bologna, 2005. ISBN 88-371-1590-3. URL `citeseer.ist.psu.edu/mascardi05languages.html`.

[15] J. J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in uml. In *IN OMG DOCUMENT AD/99-12-01. INTELLICORP INC*, pages 121–140. Springer-Verlag, 2001.

[16] A. Pokahr, L. Braubach, A. Walczak, and W. Lamersdorf. *Developing Multi-Agent Systems with JADE*, chapter Jadex - Engineering Goal-Oriented Agents, pages 254–258. Wiley and Sons, 2007.

[17] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In R. van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996. URL `citeseer.ist.psu.edu/article/rao96agentspeakl.html`.