Human-Agent Interaction, the System Level Using JASON

Antonio Chella, Francesco Lanza, and Valeria Seidita

Dip. dell'Innovazione Industriale e Digitale - Universitá degli Studi di Palermo
name.surname@unipa.it

Abstract. The main characteristic of an agent is acting on behalf of humans. Then, agents are employed as modeling paradigms for complex systems and their implementation. Today we are witnessing a growing increase in systems complexity, mainly when the presence of human beings and their interactions with the system introduces a dynamic variable not easily manageable at the design phase. Design and implementation of this type of systems highlight the problem of making the system able to decide in autonomy. In this work we propose an implementation, based on Jason, of a cognitive architecture whose modules allow structuring the decision-making process by the internal states of the agents, thus combining aspects of self-modeling and theory of the mind.

Keywords: Human-agent interaction · BDI agent · Jason.

1 Introduction

Today we want software able to cooperate with us, to anticipate our needs and to coordinate its activities with us. We also wish to have software that can autonomously and intelligently intervene and act in dynamic and changing contexts operating as humans would do. For example, a robot-human team has to cooperate to achieve a common goal in an environment not fully known. Robots and humans have to decompose the overall goal into a series of subgoals. They should then be able to understand or learn which actions are needed to reach the objective. Finally, they should match their skills with the correct steps to perform, and eventually they should delegate some task to each other.

This is a scenario concerning fully autonomous cooperative work that requires a complex software system with runtime adaptation to new situations that may lead to new requirements and constraints. Everything injected and evaluated at runtime cannot be defined during design phases, and therefore the system has to be handled as a self-adaptive system. In brief, a self-adaptive system must be aware of its objectives; it must be able to monitor the working environment and understand how far it is and if it is deviating from the objective. Moreover, it must be able to adopt alternative plans and it must also be able to generate new plans when necessary.

Important challenges in this field concern knowledge representation and updating, the selection and creation of plans at runtime, the invention of techniques

for purposefully and efficiently conveying the (runtime) decision process. These challenges lead to different solutions depending on whether we look at the architectural level or system level.

In this paper, we focus on the system level counterpart of the decision process that we achieve by employing BDI agents paradigm [7] and Jason as an agent language [5][4]. Decision processes elaborate data coming from external sources and from the environment. In many domains, it would not be enough, or it would be hard to design and implement the decision process merely employing the monitoring, analyzing, planning, acting (MAPE) cycle. In our view, decision process should take as input all the internal states of agents involved in the environment, including human. Internal states then embody the changes occurring at runtime. The project we discuss aims at considering, as a crucial part of the decision process, the data coming from the capability of attributing mental states (beliefs, desires, emotions, knowledge, abilities) to itself and the other. In brief, we take into account self-modeling and theory of mind capabilities.

Contribution and Outline of the paper. In this paper we illustrate the first steps of our ongoing work aiming at integrating self-modeling and the theory of mind in an architectural structure to implement adaptive decision process at the architectural and the system level. The architectural part extends the MAPE cycle [1] with modules allowing the perception of the external and the inner world in the form of internal states. The way we structured the architectural part and the rationale it underpins let us quickly fill the gap with the system level. We then present an extended version of the Jason reasoning cycle to map the architectural level into an agent framework.

The paper is organized as follows. Section 2 illustrates features of human-agent interaction and why we extend Jason current implementation. In Section 3 we briefly describe our architecture. Finally, Section 4 draws some discussions and conclusions.

2 Towards using BDI Agents and Jason for Implementing Human-Agent Interaction

Jason is an implementation of AgentSpeak language [9][5] that somehow allows overcoming the old denotation of software. The software is no longer something that provides a service by an exact coding, and that depends heavily on the intervention of the user. In Jason's logic, a computer program is something that has the know-how and choose actions to pursue a goal on behalf of the human and without his intervention. For this reason, a Jason program is called Agent. It does not yet have the characteristics to perfectly replicate how a human being acts, but it can autonomously process the knowledge it possesses about how to do things. So, the basic idea behind Jason is to define what is called the program know-how in the form of a set of plans. The Jason platform allows executing the deliberation process of a BDI agent that leads to choosing the intention to pursue within a set of possible states of affairs.

An agent has the ability to decide what to do, the set of actions in its repertoire to be undertaken starting from a set of data obtained through sensing and to modify the surrounding environment. In AgentSpeak, and therefore in Jason, deciding what to do means manipulating plans and the environment. Typically, a Jason agent has partial control over the environment in which it lives because it is also populated by other agents having control over their part of the environment. It can autonomously work because it is structurally defined to do this but cannot adapt itself in a dynamic environment; especially if the dynamicity of the environment derives from human and other agents interactions. The procedure for handling agent-agent interaction is standardized and mainly established at design time. Human-agent interactions have to be still explored, especially in context of cooperation between humans and agents which presupposes delegation and/or selection of actions to be undertaken even by observing the human actions and skills.

Human-agent interactions can be encoded from simple situations where everything may be identified and established at design time (environment, plans, actions and changing situations) to more complex ones where changes occur at any time and where the agent has to decide autonomously and self-adapt. It is a hard task mainly because we do not have the tools to analyze and identify all the possible elements that cause perturbation and change in the environment, so we cannot determine, at design time, a decision-making process to be implemented efficiently at the system level.

In the literature, some promising approaches [2][3] propose to solve this problem by shifting the design time to runtime. Also, some architectures containing modules for learning and memory have been introduced to pass the decision-making process through the stored and processed sensing data [10][8][6]. However, these approaches do not take into account the use of mental states, which is the primary element in our hypothesis to be able to create human-agent interaction systems behaving as human-human systems.

In the following section we briefly outline the architecture we identified to overcome the said problems, and we detail our proposal for mapping it into the system level employing the power offered by Jason.

3 Extending Jason Interpreter and its Classes

In the first part of our study, we identified an architecture heavily focused on the typical MAPE cycle. Here, some specific modules allow the decision-making process to be triggered, not only from the purely objective stimuli coming from the environment but, also, from the internal state of agents and the observation and interpretation of the actions carried out by the other agents in the working environment.

Fig. 1 shows a high-level view of the main modules of the identified architecture. The modules we added to a generic architecture centered on the sensing/plan/action cycle are highlighted in red. As can be seen, the heart of the decision-making process consists of the Reasoning module, the Action Selection

and the module for managing anticipations. The latter module represents the part of the system devoted to generating the current situation. Each time an agent has a goal to reach, it selects an action and creates anticipation of the state of the world resulting from that action. The same module receives as input the motivations, the goals and all the elements present in the memory, processes them, decides and executes the corresponding action.

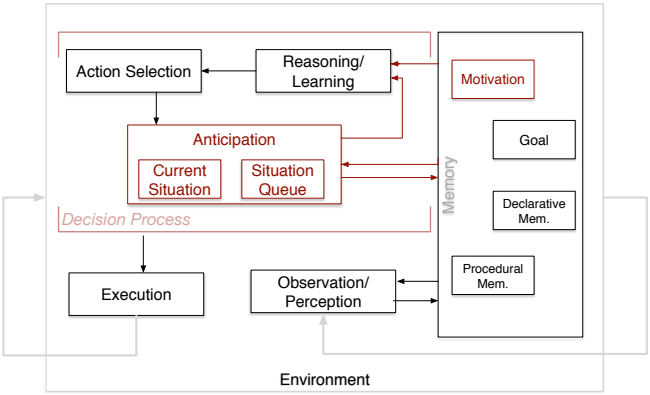

Fig. 1. The Architecture Level for Human-Agent Interaction Systems.

The Motivation module is the one triggering the anticipation and the action selection. It is the core of the decision process, here all the information and process for elaborating mental states reside and it is the module devoted to the representation of inner and outer word of each cognitive agent. Through this module, therefore, it is possible to make decisions about the actions being conveyed by the sense of self, by the ability to attribute mental states (belief, desire, intention, knowledge, capabilities) to oneself and to others and by the understanding that others have different mental states, by emotions, by the level of trust in the abilities (or more generally by trust) of others and of oneself. This architecture has been mapped onto a system by extending the Jason reasoning cycle.

The reasoning cycle of the Jason interpreter is shown in Fig. 2, it is the physical counterpart of the BDI deliberation and means-ends reasoning process. Rectangles represent the components determining the agent state; rounded boxes, diamonds and circles are used for describing the functions used in the reasoning cycle. In particular, circles model the application processes and diamonds represent the selection functions; Jason allows to modify and customize the functions represented by round boxes and diamonds. The cycle is divided into ten steps, starting from the perception of the environment to the selection of actions to be taken. The main steps of the reasoning cycle concern updating the belief base, managing the events that allow to represent the changes in the environment and in the goals, retrieving plans from the Plan Library, unifying events with plans available for the selection of the most useful plan (the so-called the applicable plan), and selecting intentions. Perception from and actions in the en-

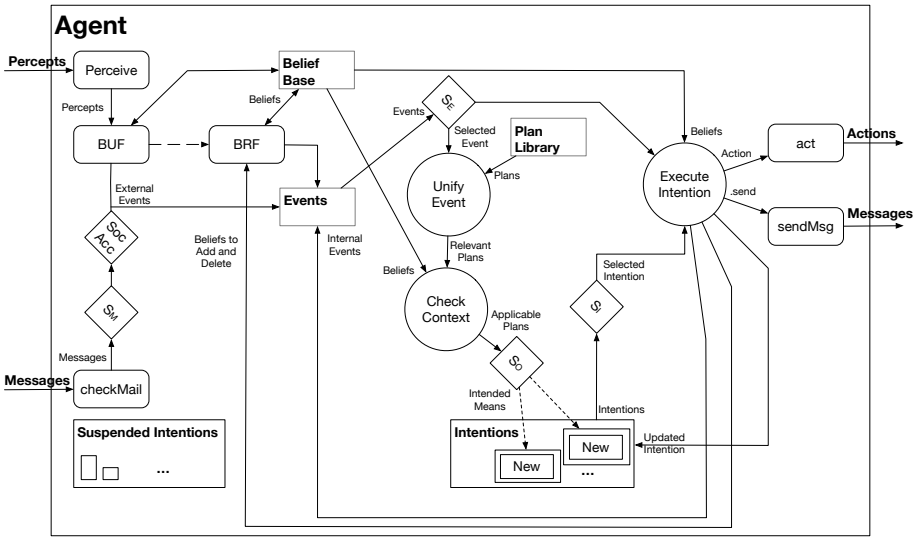

Fig. 2. Jason agent reasoning cycle. Redrawn from [5]

vironment are realized by means of the following functions: perceive, checkMail, act and sendMsg (refer to [5] for details). The cycle starts with updating the Belief Base and generating an event through the Belief Update Function (BUF) and Belief Revision Function (BRF); these functions have a correspondence with the *buf* and *brf* methods (see Fig. 4) which can be customized by users for programming more sophisticated agents. The *brf* takes the agent's current beliefs and new percepts and adds/removes beliefs. An event is then selected by the *event selection function* S_E; events represent perceived changes in the environment and in the agent's goals. Selected event is unified with the trigger event of plans retrieved from the Plan Library to determining the set of relevant plan, relevant for the given event. Once the relevant plans have been identified, they are checked against the context (a set of belief literals representing the condition for the plan to be activated) to verify whether they are logical consequence of beliefs. The result is a set of applicable plans. Given the agent's know-how expressed by the Plan Library and all the information about the environment - in the Belief Base - function *option selecion function* S_O chooses one plan that becomes the intended means handling the event by including it in the set of intentions. The Intentions components contains all the intentions ready for the execution; agent chooses employing the *intention selection function* S_I. The intention selection is then executed.

We exercised the robustness and stability of the Jason interpreter to implement BDI agents and extended the reasoning cycle to introduce the modules of the architecture above (Fig. 1). Figures 3 and 4 illustrate the components we added in the reasoning cycle for realizing the new decision process (Fig. 1) and the classes we extended and inserted in the user-defined components. Methods

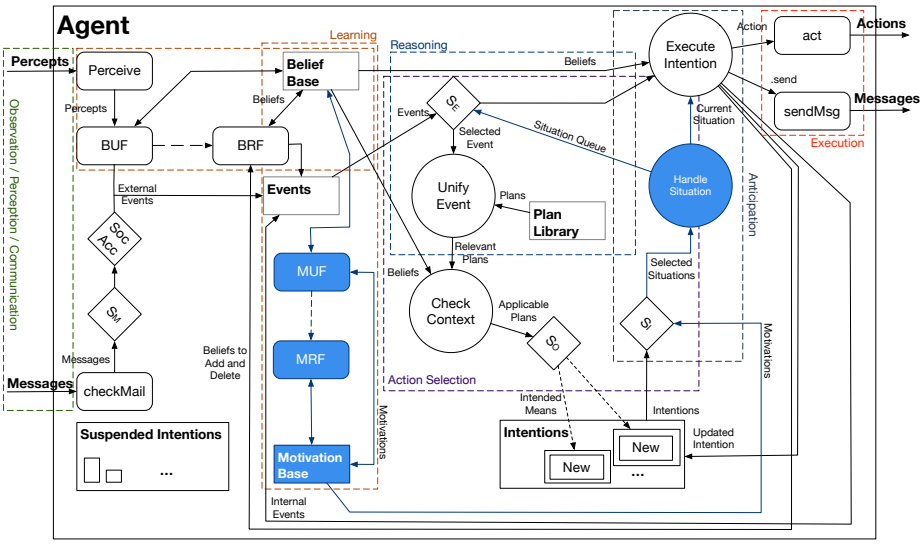

Fig. 3. Extended Jason reasoning cycle.

creation and override complete, towards the code level, the production of agents that reason by exploiting internal state.

Mainly, we introduced components and functions (in blue in the Fig.) related to the learning/reasoning module and one process realizing the newly introduced anticipation module. In parallel, concerning the Belief Base, we added the Motivation Base that includes all that beliefs related to the mental states, emotion and so on, as said before. We defined motivation as an extension of the belief to include beliefs on oneself and others; beliefs are the part of code related to the state of the world outside whereas motivations refer to the mental states. We also added Motivation Update Function (MUF) and Motivation Revision Function (MRF); at the beginning of each cycle, the first function updates and initializes all the agent's motivations and the Belief Base, takes as input a list of literals with beliefs and motivations (look at both Figures 3 and 4); in this case the input from belief base serves as it were the input from the perception. The MRF function is similar to the BRF. Motivations are elaborated from the modified S_I function; it generates a list of situations to establish which situation has to be immediately executed and the queue to be used for selecting events through S_E function. Situation is alike the state of affairs regarding the environment; situation represents the overall state of the agent also including mental states. In this way, we let agents reason on new events also generated from internal states. Finally, the process *Handle Situation* generates the current situation to be executed and provides the queue of situations to the S_E function. As regards the agent code (Fig. 4), we added a class as an extension of the BeliefBase class named Motivation. The Motivation class allows managing resources as the BeliefBase does and also querying external services to let the agent

be aware of its internal state. The core of the proposed reasoning cycle is the AgentMotivated class which extends the Agent class. Changing the selectEvent and selectIntention (S_E and S_I) function lets support the code related to MRF and MUF functions in Fig. 3 by means of *mrf* and *muf* methods. The agent invokes these methods to modify the Motivation Base. Moreover, the extension of AgArch class into AgArchMotivated lets implement the perception and action modules. Finally, what we described is the general classes architecture; as Jason prescribes, all the methods may be customized for implementing more complex agents.

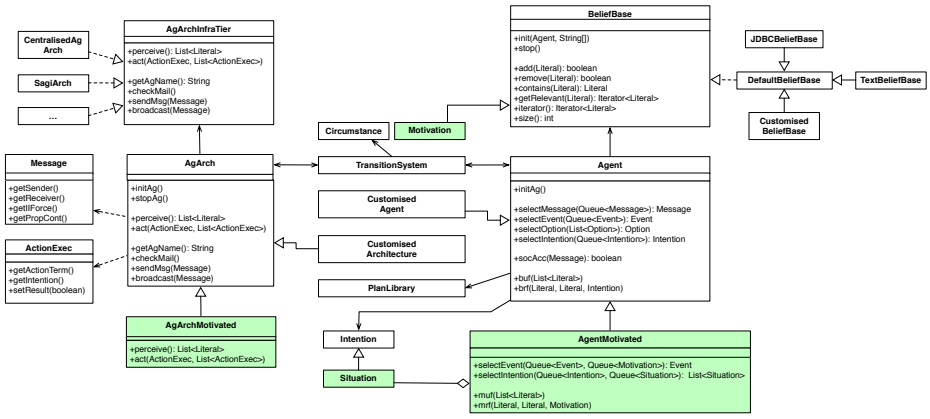

Fig. 4. Agent and Agent Architecture Class Diagram and the related extension for implementing the new reasoning cycle.

4 Discussions and Conclusions

In this paper, we present the implementation of agent's decision making process in a dynamic context. Our proposal is based on the fact that agent's decision-making-process is determined by processing data coming from observation of the external environment but also by the knowledge that the agent has about itself and the other agents acting around. The implementation of such a system is a hard task because its features can be seen only at runtime, during the interaction with the whole environment. Therefore, the system must be treated and implemented with self-adaptive characteristics.

We have exploited the power of BDI agents and the Jason language, which natively allow creating agents that perform a deliberation and means-ends reasoning process. We modified the Jason reasoning cycle to include modules to manage events, plans, and intentions selection in order to take into account what we call motivations in addition to traditional beliefs. To complete the infrastructure, even at the agent coding level, we modified classes of the Jason component

called *user-defined*. In particular, we added the classes needed to implement the part of the new reasoning cycle by adding the methods necessary for the agent to be able to choose the plan to pursue using a cognitive process based on what we called motivations that embody the mental states of the agent.

It is worth to note that the proposed cycle extension does not alter the original Jason agent reasoning at a high level, but extends its capabilities, allowing the development of agents able to manage at the same time the sense of self and the theory of the mind together with the usual decision-making process. This work is the initial part of a larger project for the implementation of complex adaptive systems including knowledge update, selection and creation of new plans at runtime. The approach we are suggesting has given some good results in the validation phase during a series of experiments conducted in the robotics laboratory of the University of Palermo. In the future, we think to definitively formalize the approach with the addition of all the design aspects to experiment with a complex case study.

Acknowledgment. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0232.

References

1. Andersson, J., Baresi, L., Bencomo, N., de Lemos, R., Gorla, A., Inverardi, P., Vogel, T.: Software engineering processes for self-adaptive systems. In: Software Engineering for Self-Adaptive Systems II, pp. 51–75. Springer (2013)
2. Baresi, L., Ghezzi, C.: The disappearing boundary between development-time and run-time. In: Proceedings of the FSE/SDP workshop on Future of software engineering research. pp. 17–22. ACM (2010)
3. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer **42**(10), 22–27 (Oct 2009). https://doi.org/10.1109/MC.2009.326
4. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. Science of Computer Programming **78**(6), 747–761 (2013)
5. Bordini, R.H., Hübner, J.F., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason, vol. 8. John Wiley & Sons (2007)
6. Franklin, S., Madl, T., D'Mello, S., Snaider, J.: Lida: A systems-level architecture for cognition, emotion, and learning. IEEE Transactions on Autonomous Mental Development **6**(1), 19–41 (2014)
7. Georgeff, M., Rao, A.: Rational software agents: from theory to practice. In: Agent technology, pp. 139–160. Springer (1998)
8. Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: An architecture for general intelligence. Artificial intelligence **33**(1), 1–64 (1987)
9. Rao, A.S.: Agentspeak (l): Bdi agents speak out in a logical computable language. In: European Workshop on Modelling Autonomous Agents in a Multi-Agent World. pp. 42–55. Springer (1996)
10. Sun, R.: The importance of cognitive architectures: An analysis based on clarion. Journal of Experimental & Theoretical Artificial Intelligence **19**(2), 159–193 (2007)