An Overview of ***Jason***

Rafael H. Bordini[1] and Jomi F. Hübner[2]

[1]Department of Computer Science
University of Durham
Durham DH1 3LE, U.K.
R.Bordini@durham.ac.uk

[2]Departamento de Sistemas e Computação
Universidade Regional de Blumenau
Blumenau, SC 89035-160, Brazil
jomi@inf.furb.br

Current trends in computer science such as the semantic web, ubiquitous computing, and self-* systems make it increasingly important that programming technology suitable for open, unpredictable, dynamic environments are made available. Many abstractions and techniques that emerged from research in multi-agent systems can have major impact in the effectiveness of (the development of) such systems. Research into agent-oriented programming languages aims at making such abstractions and techniques readily available at the level of programming languages. In this perspective, agent-oriented programming, combined with ongoing work on agent-oriented software engineering, is likely to lead to a popular new paradigm for the practical development of those complex distributed systems.

One of the most studied architectures for cognitive agents is the BDI (Beliefs-Desires-Intentions) architecture. In the area of agent-oriented programming languages in particular, AgentSpeak(L) is one of the best known languages based on the BDI architecture. AgentSpeak(L) is an abstract logic-based agent-oriented programming language introduced by Rao [8], and subsequently extended and formalised in a series of papers by Bordini, Hübner, and various colleagues. Practical BDI agents are implemented as *reactive planning systems*: they run continously, reacting to events (e.g., perceived changes in the environment) by executing plans given by the programmer. Plans are courses of actions that agents commit to execute so as achieve their goals. The pro-active behaviour of agents is possible through the notion of goals (desired states of the world) that are also part of the language in which plans are written.

Jason is a Java-based platform for the development of multi-agent systems. At the core of the platform lies an interpreter for an extended version of AgentSpeak(we use “AgentSpeak” to refer to the various extensions of the original AgentSpeak(L) language). We cannot introduce the language in this short article; a good description of the ***Jason*** language and platform can be found in [4]. Various *ad hoc* implementations of BDI-based (or “goal-based”) systems exist, but one important characteristic of AgentSpeak is its theoretical foundation; see [4] for further references. In fact, the implementation of the AgentSpeak interpreter available with ***Jason*** is directly based on the operational semantics of the language [5].

Current Features of the *Jason* Platform

The original, abstract version of the language was meant for theoretical work on relating BDI logics [9] to implementations of reactive planning systems that followed the same philosophical principles (on "practical reasoning", i.e., reasoning about how to act). The AgentSpeakextensions which ***Jason*** implements were necessary for turning the original abstract language into a practical programming language suitable for multi-agent systems. The language extensions have the following features:

Strong negation: as is well known in the ALP community, close-world assumption is not ideal for open systems where uncertainty cannot be avoided; it helps the modelling of such applications if we are able to refer to things agents believe to be true, believe to be false, or are ignorant about.

Handling of plan failures: because of the dynamic nature of typical multi-agent environments, plans can fail to achieve the goals they were written to achieve; one important aspect of reactive planning systems is that the particular choice of the specific plan to achieve a goal is left for as late as possible so as to consider the latest information the agent might have, but of course plans can still fail. ***Jason*** has a particular form of plan failure handling mechanism consisting of plans that are triggered by such failure, giving the programmer the chance to act so as to undo the effects of any action already done before the plan failed, if necessary, and then adopting the goal (that was not achieved) again, if the conditions are appropriate.

Speech-act based communication: the philosophical foundation for all the work on inter-agent communication is speech-act theory; because mental attitudes which are classically used to give semantics for speech-act based communication are formally defined for AgentSpeak we can give precise semantics for how agents interpret the basic illocutionary forces, and this has been implemented in ***Jason***. An interesting extension[1] of the language is that beliefs can have "annotations" which can be useful for application-specific tasks, but there is one standard annotations that is done automatically by ***Jason***, which is on the *source* of each particular belief. There are essentially three different types of sources of information: percepts (i.e., information obtained by sensing the environment), inter-agent communication (i.e., information obtained from other agents), and "mental notes" (i.e., beliefs added by the agent itself which can facilitate various programming tasks).

Plan annotations: in the same way that beliefs can have annotations, programmers can add annotations to plan labels, which can be used by elaborate (e.g., using decision-theoretic techniques) selection functions. Selection functions are user-defined functions which are used by the interpreter, including which plan should be

[1]Note that annotations as used here do not increase the expressive power of the language but are an elegant notation, making the belief base much more readable.

given preference in case various different plans happen to be considered applicable for a particular event.

The platform, more generally, has the following features:

Distribution: the platform makes it easy to define the agents that will take part in the system and also determine in which machines each will run, if actual distribution is necessary. The infrastructure for actual distribution can be changed (e.g., if a particular application needs to use a particular distribution platform such as JADE). Currently, two types of infrastructure are available: one that runs all agents in the same machine and another which allows distribution using SACI (`http://www.lti.pcs.usp.br/saci/`).

Environments: multi-agent systems will normally be deployed in some real-world environment. Even in that case, during development a simulation of the environment will be needed. ***Jason*** provides support for developing environments, which are programmed in Java rather than an agent language. The agent abstractions are often not appropriate for programming environments, so we provide the necessary support for this to be done in Java.

Customisation: programmers can customise two important parts of the agent platform by providing application-specific Java methods for the certain aspects of an agent and the agent architecture (note that the AgentSpeak interpreter provides only the reasoning component of the overall agent architecture). By overriding the methods of the agent class, programmers can define the selection functions (which are used by the interpreter), belief update and revision functions, as well as a "social" function which determines from which agents communication can be received. By overriding the methods of the Java class for the overall agent architecture, programmers can customise the way perception of the environment (the agent's "sensors"), inter-agent communication, and acting on the environment (the agent's "effectors") are implemented. The latter is useful, among other things because often, before deploying a multi-agent systems, programmers will want to test their system with a simulated environment. The move from simulation to real-world deployment is then done by providing the Java code that interfaces the agent's practical reasoning with the real-world environment.

Language extensibility and legacy code: the AgentSpeak extension available with ***Jason*** has a construct called "internal actions". Wherever a literal can appear in a plan, also an internal action can appear. These are then implemented in Java (or indeed any other language using JNI) as a Boolean method, and support is given, e.g., for binding of logical variables. This provides for straightforward language extensibility by user-defined internal actions, which is also a straightforward way of invoking legacy code from within the high-level agent reasoning in an elegant manner. Besides user defined internal actions, ***Jason*** comes with a library of essential standard internal actions. These implement a variety of useful operations for practical programming, but most importantly, they provide the means for programmers to do important things for BDI-inspired programming that were not possible in the original AgentSpeak language, such as checking and dropping the agent's own desires/intentions.

Integrated Development Environment: ***Jason*** is distributed with an IDE which provides a GUI for managing the system's project (the multi-agent system), editing the source code of individual agents, and running the system. Another tool provided as part of the IDE allows the user to inspect agents' internal (i.e., "mental") states when the system is running in debugging mode. The IDE is a plug-in to jEdit (`http://www.jedit.org/`), and an Eclipse plug-in is likely to be available in the future.

Ongoing Research Related to *Jason*

There is much research related to what has been done or is being done in the development of ***Jason***. Below, we mention some of this research.

Plan patterns for declarative goals: in recent work, we have devised *patterns* of AgentSpeak plans that can be used to define various types of *declarative* goals with sophisticated temporal structures. Such types of goals are quite important in the agent's literature and an essential feature of agent-oriented programming. This allows us to express for example that an agent should persist on a goal until there is evidence that it will be impossible to achieve that goal, or there is no longer any need to achieve the goal at all. The use of *patterns* for this (rather than specific language constructs) provides the same flexibility as the idea of patterns in object orientation. We are in the process of extending ***Jason*** with pre-processing to help automating the generation of those plan patterns from higher-level specifications.

Organisations: an important part of agent-oriented software engineering is related to agent *organisations*, which has received much research attention in the last few years. We are currently working on allowing specifications of agent organisations (with the related notions of roles, groups, relationships between groups, social norms, etc.) to be used in combination with ***Jason*** for programming the individual agents. The particular organisational model we use is $\mathcal{M}$oise+ [6].

Plan Exchange: Work has been done to allow plan exchange between AgentSpeak agents, which can be very useful, in particular for systems of cooperating agents, but also for application in which a large number of plans cannot be kept in the agent's plan library simultaneously (e.g., for use in PDAs with limited computational resources). The motivation for this work is this simple intuition: if you do not know how to do something, ask someone who does. However, various issues need to be considered in engineering systems where such plan exchanges can happen (e.g., which plans can be exchanged, what to do with a plan retrieved from another agent, who and when to ask for plans). This work is based on the Coo-BDI plan exchange mechanism [2].

Ontological reasoning: Although this is not available in ***Jason*** yet, in [7] it was argued that the belief base of an AgentSpeak agent should be formulated as a (populated) ontology, whereby:

1. queries to the belief base are more expressive as their results do not rely only on explicitly represented literals but can be inferred from the ontology;
2. retrieving a plan for handling an event is more flexible as it is not based solely on unification but on the subsumption relation between concepts;
3. agents may share knowledge by using ontology languages such as OWL;
4. the notion of belief update can be refined given that (ontological) consistency of a belief addition can be checked;

Concretely, in ***Jason*** we plan to use annotations to specify which ontology each belief belongs to, and use an existing tool to do the ontological reasoning when required. This further increases the need to have appropriate belief revision.

Belief revision: In ***Jason*** (and most other agent-oriented programming platforms) consistency of the belief base is left for programmers to ensure. The reason is that automatic belief revision is typically too expensive computationally, thus not appropriate for practical agent programming. In [1], a new (polynomial-time) algorithm for belief revision was introduced. The algorithm is tractable due to simplifying assumptions which are nevertheless realistic for belief bases as used in agent programming. We plan to make available an implementation of an adaptation of that algorithm for use with ***Jason***.

Final Remarks

Research in the area of agent-oriented programming languages has progressed significantly in the last few years [3], and has gone a long way since the idea was first introduced in [10]. A proportion of the many languages that have appeared in the agent programming languages literature are based on logic programming and the BDI architecture. However, there are a few aspects of such languages which might not be immediately attractive for the ALP community, in particular: (i) these language are often meant to be used for controlling the high level reasoning of an agent only, and are combined with Java programming for various aspects of a multi-agent system; and (ii) the semantics of agent languages, because of the practical reasoning aspect (e.g., plan choice and execution, and the updating of intentional structures), are normally given using operational semantics rather than the types of logical semantics so familiar to this community.

In this brief article, we have summarised the current features of ***Jason*** as well as the various strands of research related to it. ***Jason*** is distributed *Open Source* under *Gnu LGPL*; it is kindly hosted by `SourceForge.net`, and available from `http://jason.sf.net`. A book about ***Jason*** is currently being written, and is due to be published in 2007 by Wiley.

References

[1] Natasha Alechina, Mark Jago, and Brian Logan. Resource-bounded belief revision and contraction. In *Proceedings of the 3rd International Workshop on Declarative*

Agen t Languages and Technologies (DALT 2005), Utrecht, the Netherlands, July 2005.

[2] D. Ancona and V. Mascardi. Coo-BDI: Extending the BDI model with cooperativity. In *Proceedings of the First International Workshop on Declarative Agent Languages and Technologies (DALT-03)*, 2003.

[3] Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer-Verlag, 2005.

[4] Rafael H. Bordini, Jomi F. Hübner, and Renata Vieira. ***Jason*** and the golden fleece of agent-oriented programming. In Bordini et al. [3], chapter 1, pages 3–37.

[5] Rafael H. Bordini and Álvaro F. Moreira. Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L). *Annals of Mathematics and Artificial Intelligence*, 42(1–3):197–226, September 2004.

[6] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Using the $\mathcal{M}$oise+ for a cooperative framework of MAS reorganisation. In Ana L. C. Bazzan and Sofiane Labidi, editors, *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence*, volume 3171 of *Lecture Notes in Computer Science*, pages 506–515. Springer, 2004.

[7] Álvaro F. Moreira, Renata Vieira, Rafael H. Bordini, and Jomi Hübner. Agent-oriented programming with underlying ontological reasoning. In Matteo Baldoni, Ulle Endriss, Andrea Omicini, and Paolo Torroni, editors, *Proceedings of the Third International Workshop on Declarative Agent Languages and Technologies (DALT-05)*, 2005.

[8] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Walter Van de Velde and John Perram, editors, *Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, number 1038 in Lecture Notes in Artificial Intelligence, pages 42–55, London, 1996. Springer-Verlag.

[9] Anand S. Rao and Michael P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293–343, 1998.

[10] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.