Science of Computer Programming 98 (2015) 205–232

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Adding distribution and fault tolerance to Jason

Á. Fernández-Díaz *, C. Benac-Earle, L. Fredlund

Babel group, DLSIIS, Facultad de Informática, Universidad Politécnica de Madrid, Spain

HIGHLIGHTS

- We present a distribution schema for the Jason programming language.
- The monitoring technique introduced can be used to avoid fault propagation.
- The supervision mechanism presented enables fault recovery in multi-agent systems.
- The runtime system supports the extensions and can be used in a declarative way.
- Various use cases expose the potential of the extensions.

ARTICLE INFO

Article history:
Received 3 March 2013
Received in revised form 26 September 2013
Accepted 10 January 2014
Available online 23 January 2014

Keywords:
Multi-agent systems
Fault tolerance
Jason programming language
Erlang programming language
eJason

ABSTRACT

In this article we describe an extension of the multi-agent system programming language Jason with constructs for distribution and fault tolerance. This extension is completely integrated into Jason in the sense that distributing a Jason multi-agent system does not require the use of another programming language. This contrasts with the standard Java based Jason implementation, which often requires writing Java code in order to distribute Jason-based agent systems. These extensions to Jason are being implemented in eJason, an Erlang-based implementation of Jason.

We introduce two different fault tolerance mechanisms that allow fault detection and recovery. A low-level agent monitoring mechanism allows a monitoring agent to detect, and possibly recover, when another agent experiences difficulties such as e.g. hardware failures or due to network partitioning.

More novel is the second fault tolerance mechanism, *supervision*, whereby one agent acts as a supervisor to a second agent. The supervision mechanism is in addition to handling low-level faults such as the above, also capable of detecting higher-level failures such as e.g. "event overload" (an agent is incapable of timely handling all its associated events and plans) and "divergence" (an agent is not completing any iteration of its reasoning cycle). Moreover, mechanisms exist for another agent to inform a supervisor that one of its supervised agents is misbehaving.

Although these extensions are inspired by the distribution and fault tolerance mechanisms of Erlang, due to the agent perspective, the details are quite different. For instance, the supervisor mechanism of eJason is much more capable than the supervisor behaviour of Erlang, corresponding to the more abstract/higher-level perspective offered by agent-oriented programming (Jason) compared with process-oriented programming (Erlang).

As another example, from the perspective of agent programming we consider it natural to support the flexibility of the supervision trees, i.e. allow the evolution of supervision relations over time. For instance, the supervisor of an agent, as well as the supervision policy maintained for that same agent, may vary as the system evolves.

* Corresponding author.
E-mail addresses: avalor@babel.ls.fi.upm.es (Á. Fernández-Díaz), cbenac@babel.ls.fi.upm.es (C. Benac-Earle), lfredlund@babel.ls.fi.upm.es (L. Fredlund).

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2014.01.007

1. Introduction

The increasing interest in multi-agent systems (MAS) is resulting in the development of new programming languages and tools capable of supporting complex MAS development. One such language is Jason [8]. Some of the more difficult challenges faced by the multi-agent systems community, i.e., how to develop scalable and fault tolerant systems, are the same fundamental challenges that any concurrent and distributed system faces. Consequently, the agent-oriented programming languages provide mechanisms to address these issues, typically borrowing from mainstream frameworks for developing distributed systems. For instance, Jason allows the development of distributed multi-agent systems by interfacing with JADE [7,6]. However, Jason does not provide specific mechanisms to implement fault tolerant systems.

MAS and the actor model [3] have many characteristics in common. The key difference is that agents normally impose extra requirements upon the actors, typically a rational and motivational component such as the Belief-Desire-Intention architecture [22,24].

Some programming languages based on the actor model are very effective in addressing the aforesaid challenges of distributed systems. Erlang [5,9], in particular, provides excellent support for concurrency, distribution and fault tolerance. However, Erlang lacks some of the concepts, like the Belief-Desire-Intention architecture, which are relevant to the development of MAS.

In recent work [12], we presented eJason, an open source implementation of a significant subset of Jason in Erlang, with very encouraging results in terms of efficiency and scalability. Moreover, some characteristics common to Jason and Erlang (e.g. both having their syntactical roots in Prolog) made some parts of the implementation quite straightforward. However, the first eJason prototype did not permit the programming of distributed or fault tolerant multi-agent systems. A preliminary version of the distribution model and a fault tolerance mechanism for Jason were presented in [13]. In this article we build on these extensions and motivate the changes. The new extensions are being implemented in eJason, thus making it possible to develop fault tolerant distributed systems fully in Jason itself. Our up-to-date implementation of eJason and sample multi-agent systems described in this and previous documents can be downloaded at:

https://github.com/avalor/eJason.git.

The rest of the article is organised as follows: Section 2 provides background material about Erlang, Jason and eJason. Section 3 includes a brief description of platforms and techniques that address the fault tolerance of multi-agent systems. Sections 4 and 5 describe the proposed distribution model and fault tolerance mechanisms for Jason programs, respectively. Some details on the implementation in eJason of these extensions can be found in Section 6. Several examples that illustrate in detail the use of the proposed extensions are included in Section 7. Finally, Section 8 presents the conclusions and future lines of work.

2. Background

In this section we briefly introduce the background work for the contents presented in this article. Some previous knowledge of both Jason and Erlang is assumed.

2.1. Jason

Jason is an agent-oriented programming language which is an extension of AgentSpeak [21]. The standard implementation of Jason is an interpreter written in Java.

2.1.1. The Jason programming language

The Jason programming language is based on the Belief-Desire-Intention (BDI) architecture [22,24] which is highly influential on the development of multi-agent systems. The first-class constructs of the language are: beliefs, goals (desires) and plans (intentions). This approach allows the implementation of the rational part of agents by the definition of their "know-how", i.e., their knowledge about how to act in order to achieve their goals.

The Jason language also follows an environment-oriented philosophy, i.e., an agent exists in an environment which it can perceive and with which it can interact using so-called external actions. In addition, Jason allows the execution of internal actions. These actions allow the interaction with other agents (communication) or to carry out some useful tasks such as, e.g., string concatenation and printing on the standard output, among others.

2.1.2. The Java implementation of Jason

A complete description of the Java implementation of Jason can be found in [8]. This implementation of Jason allows the programming of distributed multi-agent systems by interfacing with the well-known third-party software JADE [7,6], which is compliant to FIPA recommendations [2]. JADE implements a distribution model where the agents are grouped in agent containers, which are, in turn, grouped again to compose agent platforms. These agent containers are organised hierarchically. A so-called *Main Container* exists in every platform. The Main Container provides some services to the rest of containers (e.g. maintains a global registry of all the agents in the platform) and hosts the agents that implement the Agent Management System (a white pages service) and the Directory Facilitator (a yellow pages service).