Exploring the Design Space of Agent Oriented Programming Languages

Alex Muscar

November 19, 2013

Abstract

When introduced by Shoham almost two decades ago [15], Agent Oriented Programming (AOP) was intended as a higher level alternative to Object Oriented Programming (OOP) when developing complex software systems. A lot of research has gone into AOP over the past decades, during which a vast array of approaches for developing such systems have been proposed. But the promises of the agent community have failed to materialize and agents have not gained wide acceptance. While the promise it makes is compelling—greater productivity and a gentle learning curve for novices—it still did not spark practitioners' interest. Agents are mostly regarded as experimentation tools for the Artificial Intelligence (AI) community instead of a technology for developing practical applications.

Motivated by the agent paradigm's lack of success there are voices that predict its downfall [6]. At the same time some members of the agent community are trying to raise awareness to the lack of coherence and perspectives in the community [4, 13].

We believe that the lack of success of the agent paradigm is partly due to the agent community's failure to cater to more pragmatic needs. Concurrent programming is an example of a domain in which improving the existing solution would be more than welcomed. By making concurrent computation easier to express in agent languages, a whole new area of applications would open up for AOP, from scientific applications like massively parallel simulations [17] to financial applications like *high frequency trading* [7]. We believe that by addressing more pragmatic issues, the agent community has a chance to get noticed by a broader audience.

Given the AI heritage of the agent community most of its efforts went into models of rational agents while the other aspects of agent systems have not been given the attention they deserve. There are plenty of formal models of agency and multi-agent systems to choose from, but things are somewhat bleaker when it comes to tooling, and here we are referring especially to Agent Oriented Programming Language (AOPL)s [1]. Our survey of agent oriented programming solutions supports this view: while high-level approaches are almost exclusively focused on specifying rational agents most practical approaches are implemented using generic solutions that do not have much to offer when it comes to expressivity, e.g. JADE[1].

There is also a social issue with agent languages: most of them are targeting a select group of individuals with specific needs, most of the time having non scrutable syntax, and lacking features and libraries that real world projects need. This latter aspect also impacts their adequacy as teaching languages, which would be one way of raising awareness toward the agent oriented paradigm.

Recently, the situation has started to change somewhat, and the interest in AOPLs seems to be on an ascending trend with new proposals for AOPLs [5, 11]. Also, real world applications have been developed using already established languages. More specifically the Jason language [2] has been used to develop applications ranging from

[1]Java Agent Development Framework: `http://jade.tilab.com/`

web applications [10] to mobile applications [14].

Blueprint

The premise of this thesis is that there is still place for a solution that is both high-level in the sense that it features first order concepts from the problem domain, e.g. agents, beliefs and plans, yet pragmatic and practical. It is our purpose to investigate such an approach and its implementation. Since programmers are akin to craftsmen, they can only be as productive as their tools allow them to be. Thus programming languages are a key component of developing any software system, multi-agent systems included. We will be addressing the issues discussed in this thesis from a programming language perspective.

The main contribution of this thesis is the design and implementation of Blueprint, an agent oriented programming language targeting the Common Language Runtime (CLR) framework, which emphasizes concurrency, static safety, ease of use and extensibility. Each of these aspects ads its own contribution to the related fields of AOP and AOPLs.

Before we give any further details on Blueprint's motivation and characteristics, it is worth noting that its development was something of an organic process. The ideas have germinated as an Embedded Domain Specific Language (EDSL) for agents embedded in F#, then they transformed in an extension of Jason, and finally have blossomed in the form of a standalone AOPL.

The concurrent model is an adaptation targeted towards agent languages of the one presented in [16]. This is a monadic characterization of concurrent tasks, that favours a more natural way of expressing the composition of concurrently executing tasks. A related contribution is the use of the *thread-pool* pattern for the execution of plan actions, which is in line with the current research in the AOPL world [3].

A second important contribution is the design of a statically typed agent language,

which rises the reliability of agent solutions. Again, this is in line with the current research trends in the AOPL community: only recently have solutions featuring static type systems been featured in AOPL research [12]. Keeping to the domain of static safety, Blueprint features *channel protocols*, which enforce channel communication to follow a desired pattern. While present in other languages, Blueprint is the first AOPL to feature them. The language also features asynchronous and asymmetric communication channels which act as the main synchronization points in agent programs.

Blueprint's static type system and checked communication protocols prevent a part of the errors that can appear in agents developed with existing (latently typed) agent languages. This makes programs developed in Blueprint easier to debug and extend, making the language suitable for large scale projects, as well as a teaching language. Guided by the last aspect, the language features a syntax familiar to programmers which have used languages from the C family, thus lowering the learning curve. Last, but not least, the language can easily interoperate with the underlying framework, being able to use libraries made available by the host platform.

We have also compared the performance of our implementation of the language with its main influencing solutions: Jason and JADE. As expected, Blueprint's execution mode, i.e. asynchronous plans ran in a thread pool, makes it a scalable solution when compared to JADE. While Jason is close to it—and even faster under some circumstances—it still has the problem of running out of memory when a high number of agents is started, thus making Blueprint a better solution in such situations. Overall, Blueprint is a viable solution for developing agent systems, ranging from a few agents and going up to millions of agents.

Our investigation of concurrency in AOPLs was prompted by a larger research project concerning the development of a dynamic negotiation mechanism and an accompanying framework [11, 13, 12]. We decided to use Jason for implementing an initial prototype, but given the distributed nature of our framework we were soon faced

with some of Jason's limitations. The approach we use builds on our work presented in [8, 9, 3, 5, 4]. The overall design of the BLUEPRINT language is based on some of the observations presented in [1]. The design considerations of using the *thread-pool* pattern have been analyzed in [10, 2]. We are currently investigating a possible extension of this work. Some preliminary efforts have been presented in [6].

While this work does not focus on distributed systems, concurrency is inherently present in such scenarios, so tackling this problem, even in the context of single agents or many agents running on the same machine, will benefit them as well.

Our investigation of static safety, besides the obvious motivation of having robust agent programs that can run in production systems, was prompted by our own experience teaching agent technologies, namely JADE and Jason, to undergraduate students. While written in Java, thus benefiting from its static type system, JADE does not provide any guarantees regarding message flows. One of the most common problems faced by students while developing JADE programs was that they sent messages in the wrong order—not respecting the communication protocol—ending with either deadlocked programs or with cryptic exceptions thrown by the runtime with regard to messages sent in an inappropriate order. On the other hand, the experience of teaching Jason has strengthened our opinion that static typing is a key element in writing safe programs. A second outcome of teaching Jason was that we were able to assess the impact of syntax in the adoption of a new language. Most students, who were only familiar with languages from the C family, had problems with Jason's somewhat exotic syntax. Of course this is a subjective matter, but we believe that by adopting a syntax familiar to those used with languages from the C family, we will lower the entry barrier.

We also present the sketch of a future development direction for BLUEPRINT: translating BLUEPRINT agents to JavaScript. This opens up a new array of opportunities. We believe that the field of web applications has a great potential when it comes to AOP since it is a current problem for which there is no definite solution. The myriad of pro-

gramming languages targeting the browser is a proof that this niche is still open[2]. It is our opinion that the AOP paradigm is a viable solution to developing client side applications thanks to the granular and inherently communicative nature of agents. It can also prove beneficial to the agent community as a whole, making the agent paradigm known to wider audiences.

Another beneficial point is that using the browser as a runtime, we make experimentation with the language easier, since a web browser is present on almost any computer. This could lower the entry barrier for potential users of the language. This is especially interesting in the context of teaching AOP to undergraduate students. A lot of languages have gone the same route before[3][4][5].

Contributions

The main contributions of this thesis are:

- An in-depth analysis of the main solutions for developing AOP software solutions, Jason and JADE;
- A concurrency model targeted toward AOP languages, based on the formal model presented in [16]. This is a monadic characterisation of concurrent tasks, that favours a more natural way of expressing the composition of concurrently executing tasks;
- An implementation of the previous model on top of the CLR runtime, using the *thread-pool* pattern. This approach is in line with the current research in the AOPL world [3];

[2] See http://altjs.org/ for a comprehensive list.
[3] An online Read Eval Print Loop (REPL) for Scala: http://www.simplyscala.com/
[4] An online REPL for Haskell: http://tryhaskell.org/
[5] An online REPL for F#: http://tryfsharp.org/

- The introduction of a static type system for an AOPL, which rises the reliability of agent solutions. Again, this is in line with the current research trends in the AOPL community: only recently have solutions featuring static type systems been featured in AOPL research [12];
- The introduction of *channel protocols*, which enforce channel communication to follow a desired pattern. While present in other languages, BLUEPRINT is the first AOPL to feature them;
- The use of asynchronous and asymmetric communication channels which act as the main synchronisation points in agent programs and enable fine grained *capabilities* [9, 8], thus tapping into the field of security in agent systems;
- A syntax inspired from the C family of languages, to make BLUEPRINT more accessible to new programmers, while keeping the high level concepts of *agents* and *beliefs*;
- A comparative analysis of our proposed solution with its main inspiration sources, Jason and JADE, which are at the same time the main players in the AOP world;

The Author's Publications

[1] **Muscar, Alex**. Agent Oriented Programming: from Revolution to Evolution (**DBLP**). In *ICCSW*, pages 52–58, 2011.

[2] **Muscar, Alex**. Investigating F# as a development tool for distributed multi-agent systems. 752:32–36, 2011.

[3] **Muscar, Alex**. Towards a Pragmatic Approach of Agent Development, 6th South East European Doctoral Student Conference. In *6th South East European Doctoral Student Conference*, pages 320–326, 2011.

[4] **Muscar, Alex**. Agents for the 21st Century: the Blueprint Agent Programming Language. In *Online proceedings of the 1st International Workshop on Engineering Multi-Agent Systems*, pages 255–270, 2013.

[5] **Muscar, Alex**. Extending Jason with Promises for Concurrent Computation (**Springer**, **DBLP**). In Giancarlo Fortino, Costin Bădică, Michele Malgeri, and Rainer Unland, editors, *Intelligent Distributed Computing VI*, volume 446 of *Studies in Computational Intelligence*, pages 41–50. Springer Berlin Heidelberg, 2013.

[6] **Muscar, Alex**. Join Patterns for Concurrent Agents (position paper) (**ISI**). *SCPE journal*, 43(3):181–187, October 2013.

[7] **Muscar, Alex**. Agents in the Browser – Using Agent Oriented Programming for Client Side Development (**Springer**). In Amelia Badica, Bogdan Trawinski, and Ngoc Thanh Nguyen, editors, *Recent Developments in Computational Collective Intelligence*, volume 513 of *Studies in Computational Intelligence*, pages 79–90. Springer International Publishing, 2014.

[8] **Muscar, Alex** and Costin Bădică. A Functional Approach to Agent Development: Research Agenda (**IEEE**). In *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, pages 380–385, 2011.

[9] **Muscar, Alex** and Costin Bădică. A Functional Take on Multi-Agent Systems Position Paper. *AIP Conference Proceedings*, 1389(1):865–868, 22 September 20 2011.

[10] **Muscar, Alex** and Costin Bădică. Investigating F# as a development tool for distributed multi-agent systems (*extended version*) (**IEEE**). *System Theory, Control, and Computing (ICSTCC)*, pages 1 – 6, 14-16 Oct. 2011 2011.

[11] **Muscar, Alex** and Costin Bădică. Exploring the Design Space of a Declarative Framework for Automated Negotiation: Initial Considerations (**Springer**, **DBLP**). In *AIAI (1)*, pages 264–273, 2012.

[12] **Muscar, Alex** and Costin Bădică. Towards a declarative framework for the specification of agent-driven auctions. *Engineering Intelligent Systems Journal*, 21(2/3), September 2013.

[13] **Muscar, Alex**, Laura Surcel, and Costin Bădică. Using Jason to Develop Declarative Prototypes of Automated Negotiations (**DBLP**). In *BCI (Local)*, pages 136–138, 2012.

Bibliography

[1] Costin Badica, Zoran Budimac, Hans-Dieter Burkhard, and Mirjana Ivanovic. Software agents: Languages, tools, platforms. *Comput. Sci. Inf. Syst.*, 8(2):255–298, 2011.

[2] Rafael H. Bordini, Jomi Fred Hübner, and Renata Vieira. Jason and the golden fleece of agent-oriented programming. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 3–37. Springer, 2005.

[3] Rafael C. Cardoso, Jomi Fred Hübner, and Rafael H. Bordini. Benchmarking communication in actor- and agent-based languages. In Maria L. Gini, Onn Shehory, Takayuki Ito, and Catholijn M. Jonker, editors, *AAMAS*, pages 1267–1268. IFAAMAS, 2013.

[4] Cristiano Castelfranchi. Bye-bye agents? not. *IEEE Internet Computing*, 14:93–96, March 2010.

[5] Claudia V. Grigore and Rem W. Collier. Af-raf: an agent-oriented programming language with algebraic data types. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, &*

VMIL'11, SPLASH '11 Workshops, pages 195–200, New York, NY, USA, 2011. ACM.

[6] Carl Hewitt. Perfect disruption: The paradigm shift from mental agents to orgs. *IEEE Internet Computing*, 13:90–93, January 2009.

[7] Technical Committee International Organization of Securities Commissions. Regulatory issues raised by the impact of technological changes on market integrity and efficiency. Technical report, International Organization of Securities Commissions, 2011.

[8] Mark S. Miller, Ka-Ping Yee, and Jonathan Shapiro. Capability Myths Demolished. Technical report, Systems Research Laboratory, Johns Hopkins University, 2003.

[9] Mark Samuel Miller. *Robust composition: towards a unified approach to access control and concurrency control*. PhD thesis, Johns Hopkins University, Baltimore, MD, USA, May 2006. AAI3245526.

[10] Mattia Minotti, Giulio Piancastelli, and Alessandro Ricci. Agent-oriented programming for client-side concurrent web 2.0 applications. In José Cordeiro, Joaquim Filipe, Wil Aalst, John Mylopoulos, Michael Rosemann, Michael J. Shaw, and Clemens Szyperski, editors, *Web Information Systems and Technologies*, volume 45 of *Lecture Notes in Business Information Processing*, pages 17–29. Springer Berlin Heidelberg, 2010.

[11] Alessandro Ricci and Andrea Santi. Designing a general-purpose programming language based on agent-oriented abstractions: the simpal project. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11*, SPLASH '11 Workshops, pages 159–170, New York, NY, USA, 2011. ACM.

[12] Alessandro Ricci and Andrea Santi. Typing multi-agent programs in simpal. In Mehdi Dastani, F. Hübner, Jomi, and Brian Logan, editors, *Programming Multi-Agent Systems*, volume 7837 of *Lecture Notes in Computer Science*, pages 138–157. Springer Berlin Heidelberg, 2013.

[13] Andrea Santi. From objects to agents: Rebooting agent-oriented programming for software development. *Proceedings of The 12th European Agent Systems Summer School (EASSS 2010), 2010*, 2010.

[14] Andrea Santi, Marco Guidi, and Alessandro Ricci. Jaca-android: An agent-based platform for building smart mobile applications. In Mehdi Dastani, Amal El Fallah-Seghrouchni, Jomi Hübner, and João Leite, editors, *LADS*, volume 6822 of *Lecture Notes in Computer Science*, pages 95–114. Springer, 2010.

[15] Yoav Shoham. Agent-oriented programming. *Artif. Intell.*, 60:51–92, March 1993.

[16] Don Syme, Tomas Petricek, and Dmitry Lomov. The f# asynchronous programming model. In Ricardo Rocha and John Launchbury, editors, *PADL*, volume 6539 of *Lecture Notes in Computer Science*, pages 175–189. Springer, 2011.

[17] Gaku Yamamoto, Hideki Tai, and Hideyuki Mizuta. A platform for massive agent-based simulation and its evaluation. In Nadeem Jamali, Paul Scerri, and Toshiharu Sugawara, editors, *Massively Multi-Agent Technology*, volume 5043 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2008.