From Actors to Agent-Oriented Programming Abstractions in simpAL

Alessandro Ricci
University of Bologna, Italy
a.ricci@unibo.it

Andrea Santi
University of Bologna, Italy
a.santi@unibo.it

Abstract

simpAL is a programming language introducing an agent-oriented programming abstraction layer on top of actors, with the the aim of simplifying the development of programs that need to integrate aspects related to concurrency, interaction, reactivity, distribution.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features; D.1.3 [*Programming Techniques*]: Concurrent Programming

Keywords agent-oriented programming; actors

1. Introduction – "The Free Lunch is Over"

The fundamental turn of software towards concurrency that we are witnessing in recent years – effectively summarized by the sentence *the free lunch is over* by Sutter and Larus in [10] – calls for devising effective programming abstractions that would "help build concurrent programs, just as object-oriented abstractions help build large component-based programs" in every-day programming.

Actors and Object-Oriented Concurrent Programming

To this purpose actors [2] and Object-Oriented Concurrent Programming (OOCP) [1, 4] are main references. The actor model unifies objects and concurrency and nowadays is the foundational model on which many recent languages and frameworks are based – Erlang, Scala Actors, AmbientTalk, Axum to mention some.

Looking for High-Level Programming Abstractions

Actors are a powerful, elegant yet very simple model—based solely on concurrent (re)active entities communicating by asynchronous message passing. This makes pure actor programming quite hard, in particular as soon as large concurrent and distributed programs are considered. Different kinds of ad hoc programming mechanisms have been proposed in literature to tackle specific problems–e.g., direct support for RPC-like interactions, local synchronization constrains and pattern-based receive primitives to select messages to receive, etc. Besides ad hoc mechanisms, which are often hard to combine and lack of generality, we are looking forward a single coherent programming abstraction layer, that would make as simple and natural as possible the design and development of concurrent and reactive programs.

2. Agent-Oriented Programming & the simpAL Programming Language

To that purpose, we propose a rich set of programming abstractions inspired from agents and multi-agent systems [3, 5, 9], orthogonal to the data abstraction layer. It can be defined *human-inspired* since humans and human organizations are the background metaphor. simpAL is a statically typed programming language based on these first-class abstractions [8].

A program in simpAL is modeled as an organization of agents working in a shared environment, possibly distributed into multiple workspaces. Agents model autonomous entities fulfilling pro-actively tasks, using and observing their environment and talking with other agents. The agent programming model is inspired from the BDI (Belief-Desire-Intention) model [7]. Main concepts of the model include:

- *Tasks* - description of the job to do. An agent "moves" because it has at least one task to do. Tasks are grouped into roles, defining the type of agents.
- *Plans* – define how to accomplish tasks. They are similar to procedures, but more flexible: they are composed by a set of *action rules*, specifying *when* doing *which* actions. Plans are grouped into scripts. Agents can dynamically load scripts to get the expertise to play roles and fulfill tasks.

The execution semantics of an agent is given by a *reasoning cycle*, which is an extension of actors' event loop. At every cycle, an agent first (*sense* stage) updates its *beliefs* (i.e., its

SPLASH'12, October 19–26, 2012, Tucson, Arizona, USA.
ACM 978-1-4503-1563-0/12/10.

internal state variable) about the state of the environment which is using, then (*plan* stage) it selects the actions to do in that moment using the action rules described in plans given its internal beliefs and the tasks to accomplish, and finally (*act* stage) it executes the selected actions.

The environment on the other side is used to explicitly define a shared context of agents' work. It can be programmed and modularized in terms of *artifacts*, as basic programming blocks, dual with agents. Artifact can be used to directly model non autonomous resources and tools that agents can dynamically create, share, use, part of their context [6]. They are more similar to passive objects (or, better, monitors), encapsulating:

- a set of *operations* that agents can execute (as actions). Operations are executed atomically, avoiding race conditions in the case of concurrent requests;
- a set of *observable properties*, i.e. information items that are eventually changed by actions and that are directly perceived by all the agents using such artifacts, as observers that possibly need to react to them.

3. Highlights

Like actor based approaches, simpAL allows for naturally exploiting concurrency and parallelism—agents and artifacts are logically parallel entities, mapped onto the physical OS threads and processors by the runtime. The key point with respect to actors is the programming model, which aims at being rich enough to handle the typical programming complexities arising with a flat actor model with a single uniform high-level approach.

A main example is the capability provided by the agent programming model to easily express active behaviors that need to integrate autonomy and reactive behavior [4], keeping modularity and abstraction, and to realize event-driven programming without callbacks. Another example is to recover the benefits of shared-memory models in synergy with direct communication, without exposing to the programmers (and agents) the complexity of locks and related low-level synchronization mechanisms. This is given by the environment programming model, devised to simplify the programming of artifacts functioning as coordination media, from a simple bounded buffers to tuple spaces, exploiting the basic native synchronization features of the artifact model.

Finally, a main aim of the programming model is to *keep abstractions alive from design to runtime*, making them first-class when programming, in particular. A main example in simpAL are *tasks*, which are created, assigned, manipulated by agents at runtime.

4. The simpAL Platform

simpAL comes with a Java-based platform[1] embedding a compiler, a distributed runtime infrastructure and an Eclipse-based IDE, based on xtext, including also a minimal debugger that allows for executing agents cycle by cycle, and to inspect dynamically their state as well as the artifacts' state. simpAL programs can be distributed, i.e. organization of agents distributed among workspaces running on different network nodes. The platform provides a native support to handle distribution, so that the deployment, running and debugging of distributed programs is the same of those programs that are not distributed (being all the workspaces hosted in the same node).

[1] Available as open-source project at `http://simpal.sourceforge.net`

5. Road-map

simpAL aims at being just a starting point for exploring the value of programming models based on agent-oriented abstractions. Many important issues are still to be explored, including: *(i)* formalization of the approach, by devising a proper core language/calculus – to formally define and reason about the behavior of simpALprograms; *(ii)* mechanisms for re-use and extensibility – sub-typing (e.g., defining new roles from existing roles), inheritance/composition (e.g. defining new agent scripts by composing existing plans of existing scripts); *(iii)* performance analysis and optimizations – in particular concerning the optimized execution of the agent reasoning cycle.

References

[1] G. Agha. Concurrent object-oriented programming. *Commun. ACM*, 33(9):125–141, Sept. 1990.

[2] G. A. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *J. Funct. Program.*, 7(1): 1–72, Jan. 1997.

[3] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. Special issue on multi-agent programming. *Autonomous Agents and Multi-Agent Systems*, 23 (2), 2011.

[4] J.-P. Briot, R. Guerraoui, and K.-P. Lohr. Concurrency and distribution in object-oriented programming. *ACM Comput. Surv.*, 30(3):291–329, 1998.

[5] N. R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

[6] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17 (3), Dec. 2008.

[7] A. S. Rao and M. P. Georgeff. BDI Agents: From Theory to Practice. In *First International Conference on Multi Agent Systems (ICMAS95)*, 1995.

[8] A. Ricci and A. Santi. Designing a general-purpose programming language based on agent-oriented abstractions: the simpAL project. In *Proc. of AGERE! Workshop at SPLASH*, pages 159–170, New York, NY, USA, 2011. ACM.

[9] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[10] H. Sutter and J. Larus. Software and the concurrency revolution. *ACM Queue: Tomorrow's Computing Today*, 3(7):54–62, Sept. 2005.