Rediscovering Agents for Modern SW Engineering [Position paper]

Ludo Stellingwerff
Almende B.V.
Westerstraat 50, 3016 DJ Rotterdam, The Netherlands
Email: ludo@almende.org

***Abstract*—Driven by the current technological challenges, the software engineering domain is shifting towards autonomous software structures. The architectures of tomorrow will look 'agent-like', as it has already happened with microservices. This requires a rediscovery of the autonomous agent abstraction which, when applied to modern software engineering, offers a valid solution to the issues of scalability, adaptivity and privacy.**

I. Introduction

The agent abstraction has been a success story in the scientific domain [1][2] yet the software engineering world has been reluctant to adopt it widely [3]. Nevertheless, agents are ideally suited to represent the heterogeneous, highly distributed, and dynamic systems that are ubiquitous in nowadays world. Therefore, it is plausible to claim that current technological challenges represent an opportunity for using extensively the agent abstraction in modern software development.

Currently, the most successful architecture in software engineering is the Service-Oriented Architecture (SOA), which is an evolution of the widespread client-server model [4]. Despite its success, SOA has drawbacks too, such as:

- services do not tackle satisfactorily privacy and trust concerns;
- performances of applications are highly dependent on the execution latency of the services;
- often, SOA leads to relative heavy-weight interfaces (especially when using SOAP);
- versioning and configuration management are complex, thus leading to deployment bottlenecks.

These issues are an indication of the complexity of modern software projects, which are technical cost-drivers. It is worth to mention two mitigation initiatives: the 'reactive manifesto' [5] and the development of 'microservices' [6].

This position paper focuses on the autonomous agent abstraction [7] and shows that it represents a logical next step for the type of software that reactive, microservices lead to. Importantly, it offers a more complete solution to the above mentioned challenges in SOA.

II. Autonomous agent model

Throughout the past few decades, several authors have given their own definition of 'agents' (see, for instance [8], [9], [10]). In some cases, they were placed in the field of artificial intelligence with a large focus on the cognitive, semantics aware, representation of intelligence (such as for Shoham's *agent-oriented programming model*); in other cases, the emphasis was put on autonomy (such as for Wooldrige's perspective). We follow the latter approach by proposing the following definition of an agent:

An agent is an autonomous goal-oriented representation of existing, external, sometimes abstract entities.

It is evident that the focus of this definition is mostly on the autonomous behavior of the software entities. Agents still require some form of cognitive behavior, but it is limited to a minimal level required to be 'goal-oriented'. The definition provides a requirements framework for agents from which it is possible to deduce a minimal set of capabilities an agent needs. The set described in [9] includes:

- *Autonomy*: an agent should be able to perform its actions independent of the represented entity;
- *Social ability*: an agent should be able to communicate with others, including with the entity it represents;
- *Responsiveness*: an agent should perceive its environment and react to changes in it;
- *Pro-activeness*: an agent should not just react, but take initiative, in a goal-oriented manner.

These four elements can be further translated into technical characteristics, which can be then implemented in actual software solutions:

- *Memory*: the ability to obtain a world view, keep state and remember the goals of the represented entity;
- *Communication*: the ability to communicate with other agents and the environment;
- *Control*: the ability to compare the world view with the goals and choose actions (e.g., messages) to influence the world towards those goals;
- *Self-propelled*: the ability to execute independently from external triggers.

In general, most modern software solutions contain the first two features and, depending on the application, the third one; rarely, software is self-propelled and mostly in the form of scheduled preselected tasks.

III. Example implementation

Multiple implementations can incorporate these four characteristics. As an example, this paper offers an agent model,

shown in Fig. 1, inspired by the control systems theory and hence representing an agent as a feedback system:

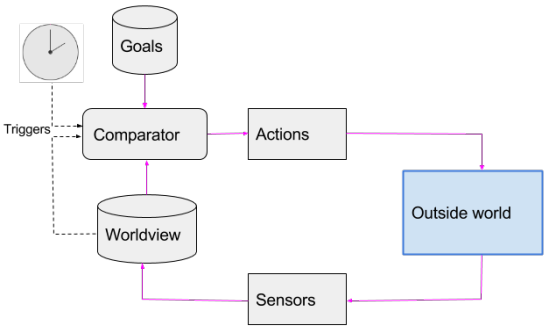

Figure 1. Feedback-based agent model.

It consists of a specific set of variables (a worldview) that get updated by sensors. Such variables are compared with a given set of goals at time-driven intervals triggered by the sensor updates themselves. The comparison can lead to the need for some action to get closer to the stated goals. The actions lead to changes in the outside world, which lead to new sensor data, updating the worldview again.

The models elements can be mapped to the four agent elements mentioned above:

- *Memory*: the worldview and goals form the memory of the system;
- *Communication*: the actions and sensors are implemented as messages;
- *Control*: the whole feedback loop forms a control system;
- *Self-propelled*: through the trigger generator this system can act proactively.

This agent model exhibits a several interesting behaviors: (i) it asynchronously interacts with the outside world; (ii) it reacts with low latency to changes in the outside world; (iii) it offers uniform handling of users, the environment and other software agents; (iv) it actively works towards system stability, even in the presence of large deviations of the goals; (v) it can cope with temporal disruptions in the sensors and/or actions.

Agents modeled in this was can work together by each working to common goals where the influence on the outside world of one agent gets seen by the other agents through the sensor data. Goals at various abstraction levels can be achieved by forming hierarchical structures of such agents.

IV. Autonomous software

Current challenges in software engineering include:

- *Privacy and trust*: possibly the biggest challenge: how to give control over data back to the subjects of that data?
- *Engineering scalability*: applications and services need to be build 'internet-scale'.
- *Developing 'adaptive' systems*: engineering models need to incorporate changing dependencies, new usage contexts and user-scripting [11];
- *Heterogeneity*: software is used in diverse environments (smart homes, robotics, big data etc.) which may involve different programming languages;
- *Dealing with legacy systems*: much of the future interaction needs to be done with currently closed environments (e.g., embedded systems, legacy business databases);
- *Robustness and decoupling*: applications become dependent on cloud services, thus causing stability bottlenecks and unwanted coupling among deployment locations.

SOA provides only a partial solution to these challenges: it naturally handles heterogeneity and access to legacy systems and, thanks to the current progress towards micro-services and reactive software, it also handles the scalability challenge. This is a good step forward, but the robustness, adaptiveness and especially trust challenges are not solved by either of them; on the contrary, the trust challenge has escalated through the involvement of SOA.

Autonomous software agents provide a different approach to these challenges, even though the agents can be build on top of the same technologies of SOA. Especially if the agents are based on web technologies, equivalent to SOA, they offer similar solutions for heterogeneity and legacy access. Additionally, the bi-directional, state-rich, self-propelled communication of agents allows handling of temporal communication and processing disturbances; their state-richness allows agents to adapt to changing context, especially if much of their logic is data-driven.

In the realm of privacy and trust, agent modelling provides a natural approach for privacy: agents are a natural container of data and its associated goals. The goals act as a filter, determining what data is needed in the world view, and more importantly, what data is *not* necessary. If the communication of data is not beneficial for some stated goal, the agent will not communicate them (or even know them altogether). This 'need-to-know' basis prevents the sort of unbridled data hoarding that some current big-data software solutions seem to focus on. However, this benefit does depend on users being in control of the goals of their own agents, which is an ethical requirement that software professionals should incorporate.

V. Discussion and future work

The direction SOA and micro-services is developing towards has a very high agent-like quality to it. Software is becoming more autonomous, highly distributed, with apps, services and dedicated (small) databases. The existing agent models can help tackle the complexities that will emerge from this future architecture. Therefore, the agent community has an important challenge: it needs to provide tools, libraries, documentation and models, aimed at handling the technical challenges in modern software engineering.

Software engineering is a highly practical, pragmatic field which will be definitely adopt a specific approach, if it is shown to be beneficial. We believe the agent based approach is worthwhile and has genuine benefits if applied to current software issues.

REFERENCES

[1] N. Jennings and M. Wooldridge. Agent-Oriented Software Engineering, Artificial Intelligence, 117:277-296, 2000.

[2] M. Calisti, F. Dignum, et al. Service-Oriented Architecture and (Multi-)Agent Systems Technology [Executive Summary], Dagstuhl Seminar Proceedings, 2010.

[3] P. Bourque and R.E. Fairley, eds., Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014; www.swebok.org

[4] M. Singh and M. Huhns, Service-Oriented Computing: Semantics, Processes, Agents. Wiley. 2005.

[5] Reactivemanifesto.org (2013). The Reactive Manifesto. [Online] Available: http://www.reactivemanifesto.org/

[6] M. Fowler and J. Lewis (2014, March 25). Microservices. [Online] Available: http://martinfowler.com/articles/microservices.html

[7] M. Wooldridge. Agent-Based Software Engineering. Unpublished seminar, first presented at Daimler-Benz, Berlin, Germany, July 1994.[Online] Available: http://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/abse.pdf

[8] S. Franklin and A. Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agent,". Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, published as Intelligent Agents III, Springer-Verlag, 1997, 21-35.

[9] N. R. Jennings and M. Wooldridge. Software Agents. IEE Review 42(1), pages 17-21. January 1996.

[10] Y. Shoham. Agent-oriented programming. Artificial Intelligence, 60(1):5192, 1993.

[11] A. Finkelstein, (2011) 10 Open Challenges in Software Engineering research challenges in software engineering, Strachey Lecture, University of Oxford. [Online] Available: http://www0.cs.ucl.ac.uk/staff/A.Finkelstein/talks/10openchall.pdf