From Objects to Agent Societies: Abstractions and Methodologies for the Engineering of Open Distributed Systems

Andrea Omicini

LIA, Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna, Viale Risorgimento 2, I-40136 Bologna, Italy
mailto:aomicini@deis.unibo.it

Abstract

We argue that the coming of the Internet era has raised issues that traditional object-oriented software engineering methodologies seem not ready to address. In this paper, we first discuss the intrinsic limitations of the object *abstraction in the engineering of complex software systems, and compare it to the* agent *abstraction. Then, we show how an agent-oriented methodology should take into account* inter-agent *aspects such as the modelling of* agent societies *and of the* agent space*, by providing engineers with specific, ad hoc abstractions and tools.*

To this purpose, we introduce the SODA *agent-oriented methodology for the analysis and design of Internet-based systems.* SODA *supplies the abstractions and procedures for engineering agent societies as well as the agent environment, including social infrastructures. The main idea in* SODA *is to exploit* coordination models, languages, *and* infrastructures *to address social issues. In particular,* SODA *shows how to choose a coordination model, how to exploit it to design social laws, how to embed them into a coordination medium, and how to build a suitable social infrastructure based on coordination services.*

1 From Object- to Agent-Oriented Methodologies

Object-oriented (OO henceforth) methodologies for the design and development of complex software systems have already proven to be effective and reliable. Generally speaking, they have shown how a suitably powerful abstraction (like the notion of object) can be fully exploited not only to define models and languages, but also to drive all the phases of the engineering of software systems.

The coming of the Internet era, however, has raised new issues that OO models, languages and methodologies seem not ready to answer. Today's applications are typically hosted by a multiplicity of distributed nodes, heterogeneous, dynamic, with no centralised control, and are often intrinsically open, in that their components are often unpredictable and unknown at design time. Moreover, the ever-increasing number of inexperienced users of Internet-based applications is making the request for intelligence in software systems inescapable.

The inadequacy of OO approaches does not derive from the methodologies, but from the object abstraction itself, which is not powerful enough to help in facing these new issues. Roughly speaking, an object is basically a reactive entity living in a closed world, where control is fully determined at the design level. Moreover, it offers no conceptual support, no natural place for embedding intelligent behaviour.

Instead, most of these issues are properly addressed by the notion of *agent* [10], which may be fruitfully thought as a methodological abstraction, rather than a totally new and revolutionary concept. Despite the many different definitions of agenthood, there is a quite common agreement on the fact that an agent is characterised by *autonomy*, *social ability*, *reactivity*, and *pro-activeness* [11].

In particular, the deliberative capability of an agent is the natural place for whichever sort of intelligence is needed, in whichever form. Pro-activeness makes it possible to abstract from control, by designing systems in terms of *tasks* and delegating *responsibility* to agents, instead of control. Moreover, the intrinsic interactive nature of agent's reactivity makes an agent a more reliable abstraction in unpredictable and dynamic environments.

Even more, the notion of agent in its full acceptation has further relevant consequences from the methodological viewpoint. In particular, social ability makes it possible to put agents together to set up *multi-agent systems* (MAS, henceforth). While each agent tries to accomplish its own task(s) by interacting with other agents and with the surrounding environment, a MAS is typically meant to pursue more complex goals than the mere sum of its agents' goals. This is typically done by organising agents into *societies*,

defining or ruling their mutual interaction so as to produce useful global behaviours. Agent societies can themselves be exploited as independent, first-class abstractions in the engineering of complex software systems.

Finally, when looking at agents as *situated entities*, which cannot be thought separately from the environment they live in, the idea of modelling a software system as a MAS without modelling the agents' *environment* seems basically ineffective. Agents and societies live in environments that may be heterogeneous, dynamic, open, distributed and unpredictable, like the Internet is. Such features cannot have but a deep influence on the way in which software systems are built, when thought as MAS: so, agenthood calls for new abstractions modelling the agent's environment, in terms of resources, services, topology, and so on.

2 Interaction as a Dimension for Software Engineering Methodologies

By adopting an agent-oriented viewpoint, three issues emerge that are not covered by traditional OO methodologies: *agenthood*, *societies*, and *environment*.

The first issue basically involves intra-agent aspects, that is, the analysis, design and development of individual agents. This is basically a *computational* issue [9], and agent-oriented engineering has been mainly concerned with this aspects, till now. Instead, the two latter issues concern the *interaction* of agents with their environment, the resources, and other agents, which is instead not so well-addressed by the existing literature on agent-oriented software engineering. By taking interaction as an independent dimension for the analysis, design and development of MAS, the point is then how such a dimension should affect methodologies for the engineering of complex software systems when they are built as MAS.

2.1 Society

Agents are individuals entities with social abilities [11]. In general, they have a partial representation of the world around them, a limited ability to sense and change it, and typically rely on other agents for anything falling outside of their scope or reach. So, agents have typically to be thought as living dipped into societies: the behaviour of an individual agent is usually not understandable if not in relation to a social structure. The behaviour of a buyer agent in an auction is difficult to be explained out of the context of the auction itself and of the rules which govern it. Dually, the behaviour of a society of agents cannot generally be expressed in terms of the behaviour of its composing agents. So, the rules governing an auction, in conjunction with the behaviour of the individual agents participating to it, lead to a global behaviour which could not be reduced to the mere composition of the individual's behaviour [9]. Social rules harness inter-agent interaction, and drive the global behaviour of a society as an independent entity of a multi-agent system.

So, societies should not be conceived and built as the mere composition of a number of agents, separately engineered. Instead, agent-oriented methodologies should take into account agent societies as first-class entities, to be exploited in the analysis, design and development of complex software systems, and engineered as such. For this purpose, agent-oriented methodologies should provide for specific models, abstractions and technologies for engineering agent societies. In particular, a methodology should drive engineers in understanding which societies are required, which social laws they need, how social rules should be designed, and where they should be embedded. For instance, one should be able to determine how much of a social behaviour should be embodied in agents, and how much should be instead charged upon social infrastructures – a particularly relevant issue when open systems are concerned.

This is where *coordination models* [8] come at hand: in fact, *coordination media* – that is, the abstractions mediating agent interaction – make it possible to design social laws, and to deploy them in terms of *coordination rules*. As discussed in [5, 2], a coordination medium can work as the core of an agent society, and its behaviour can be designed so as to build the desired social behaviour. So, the choice of a coordination model is an essential step in the analysis and design of multi-agent systems, and of agent societies in particular. Correspondingly, we argue that any effective agent-oriented methodology should drive engineers in the choice of the most effective coordination model and language to shape the agent interaction space, and in particular to build up social behaviour.

2.2 Environment

When looking at agents as *situated entities*, which cannot be thought separately from the environment they live in, the idea of modelling a software system as a multi-agent system without modelling the agent *environment* seems to be ineffective from its very ground. Generally speaking, agents and societies live in environments which may be heterogeneous, dynamic, open, distributed, and unpredictable – like the Internet is. Obviously, the design of agents and societies cannot but take these properties into account, since they affect the way in which they represent the world they live in, and how they plan and deliberate their course of actions within it. Correspondingly, agent-oriented methodologies should allow the agent space to be modelled, by making it possible to determine from the very beginning of the engineering process how the properties of the agent environment

affect the engineering of agents and societies.

Even more, it is often the case that the agent space itself is subject to engineering – in particular, whenever it is possible and useful to factorise properties of the systems and to delegate them to the *infrastructure*. One may think for instance of directory services, shared knowledge bases, authentication services, and so on: the way in which they are built and made available to the agents of a multi-agent system obviously affects the way in which the system and its components are engineered. So, agent-oriented methodologies should help not only in modelling the agent environment, but also in building it.

3 SODA

SODA (Societies in Open and Distributed Agent spaces) is a methodology for the analysis and design of Internet-based applications as multi-agent systems. The goal of SODA is to define a coherent conceptual framework and a comprehensive software engineering procedure which accounts for the analysis and design of individual agents, agent societies, and agent environments. SODA is not meant to account for intra-agent issues: designing a multi-agent system with SODA leads to defining agents in terms of their required observable behaviour, and of the role they play in the multi-agent system. Then, whichever methodology one may choose to define the agent structure and inner functionality, it can easily used to complete the SODA one.

Instead, SODA concentrates on the inter-agent issues, like the engineering of societies and infrastructures for multi-agent systems. Since this conceptually covers all the interactions within an agent system, the design phase of SODA deeply relies on the notion of *coordination model* [8]. In particular, coordination models and languages are taken as the sources for the abstractions and mechanisms required to engineer societies: social rules are designed as coordination laws, and social infrastructures are build upon coordination systems.

3.1 Analysis

During the analysis phase, the application domain is studied and modelled, the available computational resources and the technological constraints are listed, the fundamental application goals and targets are devised out. The result of the analysis phase is typically expressed in terms of high-level abstractions and of their mutual relationships, providing designers with a formal or semi-formal description of the intended overall application structure and organisation.

Since by definition agents have goals that they pursue pro-actively, agent-oriented analysis can rely on agent's *responsibility* to carry on one or more *tasks*. Furthermore, agents live dipped into an environment, which may be distributed, heterogeneous, unpredictable and dynamic. So, the analysis phase should explicitly take into account and model the required and desired features of the agent application environment, by modelling it in terms of the *services* made available to agents. Finally, since agents are basically interactive entities, depending on other agents and available resources to accomplish their tasks, the analysis phase should explicitly model the interaction protocols in terms of the information required and produced by agents and resources.

So, the SODA analysis phase focuses on three distinct models:

- the *role model* – the application goals are modelled in terms of the *tasks* to be achieved, which are associated to *roles* and *groups*
- the *resource model* – the application environment is modelled in terms of the available *services*, which are associated to abstract *resources*
- the *interaction model* – the interaction involving roles, groups and resources is modelled in terms *interaction protocols*, expressed as *information* required and produced by roles and resources, and *interaction rules*, governing interaction within groups

Even though conceptually different, these three models used by SODA as the basis for the analysis phase are strictly related, and have to be defined in a consistent way.

3.1.1 The role model

Tasks are expressed in terms of the *responsibilities* they involve, of the *competences* they require, and of the *resources* they depend upon. Responsibilities are expressed in terms of the state(s) of the world which has to result from the task's accomplishment.

Tasks are classified as either *individual* or *social* ones. Typically, social tasks require a number of different competences, and the access to several different resources, whereas individual tasks are more likely to require well-delimited competence and limited resources (see [2] for an example).

Each individual task is associated to an individual *role*, which is by consequence first defined in terms of the responsibilities it carries. Analogously, social tasks are assigned to *groups*. Groups are defined in terms of both the responsibility related to their social task, and the *social roles* participating in the group. A social role describes the role played by an individual within a group, and may either coincide with an already defined (individual) role, or be defined *ex-novo*, in the same form as an individual one, by specifying its task as a sub-task of its group's one.

3.1.2 The resource model

Services express functionalities provided by the agent environment to an agent system – like recording an information, querying a sensor, verifying an identity. In this phase, each service is associated to an abstract *resource*, which is then first defined in terms of the service it provides.

Each resource defines abstract *access modes*, modelling the different ways in which the service it provides can be exploited by agents. If a task assigned to a role or a group call for a given service, the access modes required have to be determined and expressed in terms of *permission* to access the resource providing for that service. Such a permission is then associated to that role or group.

3.1.3 The interaction model

Analysing the interaction model in SODA amounts to the definition of *interaction protocols*, for roles and resources, and *interaction rules*, for groups.

An interaction protocol associated to a role is defined in terms of the information required and produced by the role in order to accomplish its individual task. An interaction protocol associated to a resource is defined in terms of the information required to invoke the service provided by the resource itself, and by the information returned when the invoked service has been brought to an end, either successfully or not. An interaction rule is instead associated to a group, and governs the interactions among social roles and resources so as to make the group accomplish its social task.

It is worth to be noticed the degree of uncoupling provided by this approach: each interaction protocol is not specifically bounded to any other, and can be defined somehow independently – simply requiring the specification of the information needed, but not its source. Obviously, the final outcome of the analysis phase should account for this, too, by ensuring that for any information required by any protocol, there is an entity in the system in charge of providing it.

3.1.4 The outcome

In all, the results of the SODA analysis phase are expressed in terms of roles, groups and resources. To summarise,

- a role is defined in terms of its individual task, its permissions to access the resources, and its corresponding interaction protocol
- a group is defined in terms of its social roles, its social task, its permissions to access the resources, and its corresponding interaction rule
- a resource is defined in terms of the service it provides, its access modes, the permissions granted to roles and groups to exploit its service, and its corresponding interaction protocol

3.2 Design

Design is concerned with the representation of the abstract models resulting from the analysis phase in terms of the design abstractions provided by the methodology. Differently from the analysis phase, a satisfactory result of the design phases is typically expressed in terms of abstractions that can be mapped one-to-one on to the actual components of the final system.

The SODA's design phase aims at defining three strictly related models:

- the *agent model* – individual and social roles are mapped upon *agent* classes
- the *society model* – groups are mapped onto *societies of agents*, to be designed around *coordination abstractions*
- the *environment model* – resources are mapped onto *infrastructure* classes, and associated to *topological abstractions*

3.2.1 The agent model

An *agent class* is defined as a set of (one or more) roles, both individual and social ones. So, an agent class is first characterised by the tasks, the set of the permissions, and the interaction protocols associated to its roles. Agent classes can be further characterised in terms of other features: their *cardinality* (the number of agents of that class), their *location* (with respect to the topological model defined in this phase – either fixed, for static agents, or variable, for mobile agents), their *source* (from inside or outside the system, given the assumption of openness).

The design of the agents of a class should account for all the specifications coming from the SODA analysis phase – but may exploit in principle any other methodology for the design of individual agents, since these issue is not covered by SODA. What is determined by SODA is the outcome of this phase, that is, the *observable behaviour* of the agent in terms of all its interactions with its surrounding environment. The observable behaviour is defined by the interaction protocols, delimited by the permission sets, and finalised to the achievement of the agent's tasks.

3.2.2 The society model

Each group is mapped onto a society of agents. So, an agent society is first characterised by the social roles, the social tasks, the set of the permissions, and the interaction rules associated to its groups.

Since the agent model assigns also social roles to agents, the main issue in the society model is how to design interactions rules so as to make societies accomplish their social tasks. Being a problem of managing agent interaction, harnessing it to achieve the desired social behaviour, this is basically a coordination issue [7]. As a result, societies in SODA are designed around *coordination media*, that is, the abstractions provided by coordination models for the coordination of multi-component systems [1].

So, the first point in the society design is the choice of the fittest coordination model – that is, the one providing abstractions that are expressive enough to model the society's interaction rules [4]. Thus, a society is designed around coordination media [5], which embody the interaction rules of its groups in terms of *coordination rules*. The behaviour of the suitably-designed coordination media, along with the behaviour of the agents playing social roles and interacting through such media, makes an agent society pursue its social tasks as a whole. This allows societies of agents to be designed as first-class entities, as shown in [2] where an example is also discussed.

3.2.3 The environment model

Resources are mapped onto infrastructure classes. So, an infrastructure class is first characterised by the services, the access modes, the permissions granted to roles and groups, and the interaction protocols associated to its resources. Infrastructure classes can be further characterised in terms of other features: their *cardinality* (the number of infrastructure components belonging to that class), their *location* (with respect to topological abstractions), their *owner* (which may be or not the same as the one of the agent system, given the decentralised control assumption).

The design of the components belonging to an infrastructure class may follow the most appropriate methodology for that class – databases, expert systems, security facilities, all can be developed according to the most suited specific methodology, since SODA does not address these issues specifically. Again, what is determined by SODA is the outcome of this phase, that is, the services to be provided by each infrastructure component, and its *interfaces*, as resulting from its associated interaction protocols.

Finally, SODA assumes that a topological model of the agent environment is provided by the designer – but does not provide for topological abstractions by its own, since any system, any application domain may call for different approaches to this problem. However, as an example of an expressive set of topological abstractions that may easily fit many Internet-based multi-agent systems, one may look to *places*, *domains* and *gateways* as defined by the TuCSoN model for the coordination of Internet agents [3].

4 Conclusions and further work

The main reference for the development of the SODA ideas is the pioneering work on Gaia [12]. Gaia, to our knowledge, is the first agent-oriented software engineering methodology that explicitly takes into account societies (there, mainly referred to as *organisations*) as first-class entities, by providing a coherent conceptual framework for the analysis and design of multi-agent systems. Even though at an early stage of its development, SODA addresses some of the Gaia shortcomings, which does not suit well open systems, and cannot easily deal with self-interested agents. In SODA, this problems are overcome by exploiting suitable coordination models as the basis for the engineering of societies, enabling open societies to be designed around suitably-designed coordination media, and social rules to be designed and enforced in terms of coordination rules.

In addition, SODA is the first agent-oriented methodology to our knowledge to explicitly take the agent space into account, and to provide engineers with specific abstractions and procedures for the design of agent infrastructures.

Early versions of the SODA methodology have already been used for the analysis and design of Internet-based multi-agent systems [6, 5]: however, the methodology was neither explicitly formalised nor named before. In the near future, we intend to exploit SODA in the design of real Internet-based multi-agent systems, so as to further verify its effectiveness at the current stage of its development.

References

[1] Paolo Ciancarini. Coordination models and languages as software integrators. *ACM Computing Surveys*, 28(2):300–302, June 1996.

[2] Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli. Multiagent system engineering: the coordination viewpoint. In Nicholas R. Jennings and Yves Lespérance, editors, *Intelligent Agents VI — Proceedings of the 6th International Workshop on Agent Theories, Architectures, and Languages (ATAL'99)*, volume 1767 of *LNAI*. Springer-Verlag, February 2000. Invited paper.

[3] Marco Cremonini, Andrea Omicini, and Franco Zambonelli. Multi-agent systems on the Internet: Extending the scope of coordination towards security and topology. In Francisco J. Garijo and Magnus Boman, editors, *Multi-Agent Systems Engineering – Proceedings of the 9th European Workshop on Modelling Autonoumous Agents in a Multi-Agent World¡ (MAMAAW'99)*, volume 1647 of *LNAI*, pages 77–88. Springer-Verlag, June 30 - July 2 1999.

[4] Enrico Denti, Antonio Natali, and Andrea Omicini. On the expressive power of a language for programming coordination media. In *Proceedings of the 1998 ACM Symposium on Applied Computing (SAC'98)*, pages 169–177. ACM, February 27 - March 1 1998. Track on Coordination Models, Languages and Applications.

[5] Enrico Denti and Andrea Omicini. Designing multi-agent systems around a programmable communication abstraction. In John-Jules Ch. Meyer and Pierre-Yves Schobbens, editors, *Formal Models of Agents – ESPRIT Project ModelAge Final Report*, volume 1760 of *LNAI*, pages 90–102. Springer-Verlag, 1999.

[6] Enrico Denti and Andrea Omicini. Engineering multi-agent systems in LuCe. In Stephen Rochefort, Fariba Sadri, and Francesca Toni, editors, *Proceedings of the ICLP'99 International Workshop on Multi-Agent Systems in Logic Programming (MAS'99)*, Las Cruces (NM), November 30 1999.

[7] Thomas Malone and Kevin Crowstone. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[8] George A. Papadopoulos and Farhad Arbab. Coordination models and languages. *Advances in Computers*, 46:The Engineering of Large Systems:329–400, August 1998.

[9] Peter Wegner. Why interaction is more powerful than computing. *Communications of the ACM*, 40(5):80–91, May 1997.

[10] Michael J. Wooldridge. Agent-based software engineering. *IEE Proceedings on Software Engineering*, 144(1):26–37, February 1997.

[11] Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[12] Michael J. Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 2000. To appear.