SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems

Andrea Omicini

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

Abstract. The notion of society should play a central role in agent-oriented software engineering as a first-class abstraction around which complex systems can be designed and built as multi-agent systems. We argue that an effective agent-oriented methodology should account for *inter-agent* aspects by providing engineers with specific abstractions and tools for the analysis and design of *agent societies* and *agent environments*.
In this paper, we outline the SODA agent-oriented methodology for the analysis and design of Internet-based systems. Based on the core notion of *task*, SODA promotes the separation of individual and social issues, and focuses on the social aspects of agent-oriented software engineering. In particular, SODA allow the agent environment to be explicitly modelled and mapped onto suitably-defined agent infrastructures.

1 Introduction

The engineering of complex applications in the Internet era raises new problems which require new models, languages and methodologies. Agent-based approaches [21] exploit the agent abstraction to address issues like distribution, heterogeneity, decentralisation of control, unpredictability, and need for intelligence [22]. Agents situatedness and their reactivity help to deal with dynamic and unpredictable environments; their pro-activeness in pursuing goals makes it possible to abstract away from the control issue and to easily deal with decentralisation of control; and so on.

However, the most mature approaches to agent-oriented engineering have till now concentrated on *intra-agent* aspects – how to build an individual agent, starting from *ad hoc* agent languages, architectures, and methodologies. These approaches implicitly promote methodologies for the engineering of multi-agent systems where systems are built as a sum of separately engineered agent components, which are then put together by exploiting some technology or infrastructure for interoperability – like ACL, mediation services, brokers, and so on.

According to the most recent research trends, this neglects one of the most relevant aspects of agent-based systems, that is, the social ones [9, 18, 19]. Agents are not simple software components to be first built, then combined: they are goal-driven individuals, who assume to live and interact with other individuals within a *society*. In the same way as human ones, agent societies exhibit global behaviours which cannot be reduced to the mere sum of the behaviours of their individual components. As a result, societies should be considered as first-class components of multi-agent systems, and specific models, abstractions, languages, and methodologies have to be provided for their engineering [11, 13, 14, 25].

Even more, the *agent environment*, that is, the space where agents live and interact, is not neutral with respect to system design and development. Building a MAS in an open, distributed, decentralised, heterogeneous, dynamic, and unpredictable environment obviously affects the way in which such a system is conceived and deployed. As a peculiar example, think of an open system where some resources have to be made available to explorer agents coming from unknown sources – like buyer agents, for instance. The engineering of such a system would simply amount to designing the agent environment in terms of available resources and services, and deploying a suitably-configured infrastructure – possibly, without writing a single line of agent code. As a result, also the structure of the agent environment should be adequately modelled through specific abstractions, and taken into account at every step of the engineering process of a multi-agent system.

In this context, this paper outlines the SODA agent-oriented methodology for the analysis and design of Internet-based systems, aimed at defining abstractions and procedures specifically tailored to the engineering of agent societies and environments. Based on primitive notion of *task*, SODA promotes the separation of individual and social issues since the very early analysis phase. Since it intentionally does not address intra-agent issues, SODA is not a complete methodology, and focuses instead on the social aspects of agent-oriented software engineering, by exploiting *coordination models* and *technologies* [15]. In particular, SODA allow the agent environment to be explicitly modelled and mapped onto suitably-defined agent infrastructures.

2 Society and environment in agent systems

Till now, agent-oriented engineering [21] has been mainly concerned with intra-agent aspects, that is, the analysis, design and development of individual agents. This is basically a *computational* issue [20], which involves the way in which each agent works when seen as an individual (software) system.

As suggested by many recent research efforts [9, 18, 19], inter-agent issues should instead be considered at least as relevant as intra-agent one, and handled as that. In particular, by taking *interaction* as an independent dimension for the analysis, design and development of multi-agent system, it should be made clear how such a dimension should affect the methodologies for the engineering of complex software systems as a multi-agent one [10]. For instance, Miles, Joy, and Luck [13] present a methodology for agent-oriented software engineering based on the analysis of agent interaction.

Agents in a multi-agent system interact by living and working within their environment, and by relating with other agents. Correspondingly, inter-agent aspects in multi-agent systems basically amount to two strongly related issues: the social and the environment one.

2.1 Society

Agents are individual entities with social abilities [22]. 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 are to be thought as living dipped into societies: the behaviour of an individual agent is often not understandable outside its 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 that 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 that could not be reduced to the mere composition of the individual's behaviour [20]. Social rules harness agent interaction, and drive the global behaviour of a society towards the accomplishment of its global goals.

So, societies should be no longer built by merely combining a number of separately engineered agents. Instead, agent-oriented methodologies should adopt agent societies as first-class abstractions to be exploited in the analysis, design, and development of complex software systems. For this purpose, agent-oriented methodologies should supply specific models, abstractions, and technologies for the engineering of agent societies. In particular, a methodology should help engineers to determine the social structures required, the social laws they need, how social rules should be designed, and how they should be enforced. 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.

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 that may be heterogeneous, dynamic, open, distributed, and unpredictable – like the Internet. The properties of the environment obviously affect the way in which agents represent the world they live in, and how they plan and deliberate their course of actions. So, agent-oriented methodologies should make it possible to model the agent environment from the earliest phases of the engineering process, and to express dependencies within the agents and the environment itself.

Even more, the features of the agent environment are often not completely predetermined, but may be partially defined according to the systems needs. So, the environment of a multi-agent system may be subject to an engineering process, aimed at shaping and configuring the environment itself. For instance, one may think of directory services, shared knowledge bases, authentication services, and so on: how they are built and made available to the agents of a multi-agent system both affects and depends on the way in which the system and its agents are engineered. So, agent-oriented methodologies should help not only to model the agent environment, but also to shape and build 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 that accounts for the analysis and design of individual agents, agent societies, and agent environments. SODA is not concerned with intra-agent issues: designing a multi-agent system with SODA leads to define agents in terms of their required observable behaviour and their role in the multi-agent system. Then, whichever methodology one may choose to define the agent structure and inner functionality, it could be easily used in conjunction with SODA.

Instead, SODA concentrates on 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* [2, 16, 17]. In particular, as discussed in [4, 7], coordination models and languages are taken as the sources of the abstractions and mechanisms required to engineer agent societies: social rules are designed as coordination laws and embedded into coordination media, and social infrastructures are built 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 pointed out. The result of the analysis phase is typically expressed in terms of high-level abstractions and 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 *responsibility* to carry on one or more *tasks*. Furthermore, agents live dipped into an environment, which may be distributed, heterogeneous, dynamic, and unpredictable. 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 required *resources* and the *services* made available to agents. Finally, since agents are basically interactive entities, which depend on other agents and available resources to pursue their tasks, the analysis phase should explicitly model the interaction protocols in terms of the information required and provided by agents and resources.

So, the SODA analysis phase exploits three different 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 *services* available, which are associated to abstract *resources*
- the *interaction model* – the interaction involving roles, groups and resources is modelled in terms of *interaction protocols*, expressed as *information* required and provided by roles and resources, and *interaction rules*, governing interaction within groups.

The above models represent the basis of the SODA analysis phase. Even though conceptually distinct, they are obviously strictly related, and should be defined in a consistent way.

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 that should result from the task accomplishment.

Tasks are classified as either *individual* or *social* ones. Typically, social tasks are those that require a number of different competences, and the access to several different resources, whereas individual ones are more likely to require well-delimited competence and limited resources (see [4] for an example).

Each individual task is associated to an individual *role*, which by consequence is 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.

The resource model *Services* express functionalities provided by the agent environment to a multi-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 firstly 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 requires a given service, the access modes are determined and expressed in terms of the granted *permission* to access the resource in charge of that service. Such a permission is then associated to that role or group.

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 provided 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 noted that this approach ensures a form of uncoupling: each interaction protocol is not specifically bounded to any other, and can be defined somehow independently – by 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 at least one entity in the system in charge of supplying such information.

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 the corresponding interaction protocol
- a group is defined in terms of its social task, its permissions to access the resources, the participating social roles, and the 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 the 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 onto the actual components of the deployed system.

The SODA design phase is based on 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, which are designed and organised around *coordination abstractions*
- the *environment model* – resources are mapped onto *infrastructure* classes, and associated to *topological abstractions*.

The agent model An *agent class* is defined as a set of (one or more) roles, both individual and social ones. As a result, 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 this 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 the surrounding environment. Such a behaviour is defined by the interaction protocols, delimited by the permission sets, and finalised to the achievement of the agent tasks.

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

The agent model also assigns social roles to agents, so that the main issue in the society model is how to design interaction rules so as to make societies to accomplish their social tasks. Since it deals with managing agent interaction, the problem of achieving the desired social behaviour by means of suitable social rules is basically a *coordination* issue [12]. 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 [3].

So, the first point in the design of agent societies is the choice of the fittest coordination model – that is, the one providing the abstractions that are expressive enough to model the society interaction rules [6]. Thus, a society is designed around coordination media [7] embodying 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 [4] where an example is also discussed.

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 assumption of decentralised control).

The design of the components belonging to an infrastructure class may follow the most appropriate methodology for that class – since SODA does not specifically address these issues, components like databases, expert systems, or security facilities, can all be developed according to the most suited specific methodology. 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 and 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 [5].

The outcome In all, the results of the SODA design phase are expressed in terms of agent classes, societies of agents, and infrastructure classes. To summarise,

– an agent class is defined in terms of its individual and social roles, as well as its cardinality, location, and source

- a society of agents is defined in terms of its groups, as well as its corresponding coordination abstraction(s)
- an infrastructure class is defined in terms of its resources, as well as its cardinality, location, and owner.

4 Related works and conclusions

The main reference for the development of SODA is represented by the pioneering work on Gaia [23]. 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 shortcomings of Gaia, which does not suit well open systems, and cannot easily deal with self-interested agents [24]. In addition, SODA is the first agent-oriented methodology to our knowledge to explicitly take the agent environment into account, and provide engineers with specific abstractions and procedures for the design of agent infrastructures.

Zambonelli, Jennings, and Wooldridge [25] also try to address Gaia shortcomings, by putting the notion of organisation at the core of their agent-oriented methodology. A similar approach is proposed by Kendall [11], which adopts *role models* as the main organisational abstraction for modelling multi-agent systems. There, however, the notion of role is taken as primitive, whereas SODA considers role as a derived notion, and task and service as primitive ones. In turn, Blanzieri and Giorgini [1] address the openness issue by proposing a conceptual infrastructure based on the notion of *implicit culture*.

Early versions of the SODA methodology have already been used for the analysis and design of Internet-based multi-agent systems [7, 8]: however, the methodology was never explicitly neither 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.

References

1. Enrico Blanzieri and Paolo Giorgini. Implicit culture and multi-agent systems. In this volume.
2. Nadia Busi, Paolo Ciancarini, Roberto Gorrieri, and Gianluigi Zavattaro. *Coordination Models: A Guided Tour*, chapter 1. In Omicini et al. [15], December 2000.
3. Paolo Ciancarini. Coordination models and languages as software integrators. *ACM Computing Surveys*, 28(2):300–302, June 1996.
4. Paolo Ciancarini, Andrea Omicini, and Franco Zambonelli. Multiagent system engineering: the coordination viewpoint. In Nicholas R. Jennings and Yves Lespérance, editors, *Intelligent Agents VI — Agent Theories, Architectures, and Languages*, volume 1767 of *LNAI*, pages 250–259. Springer-Verlag, February 2000.
5. 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.

6. 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.
7. Enrico Denti and Andrea Omicini. Designing multi-agent systems around an extensible 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.
8. 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.
9. Fumio Hattori, Takeshi Ohguro, Makoto Yokoo, Shigeo Matsubara, and Sen Yoshida. Socialware: Multiagent systems for supporting network communities. *Communications of the ACM*, 42(3):55–61, March 1999. Special Section on Multiagent Systems on the Net.
10. Michael N. Huhns. Interaction-oriented programming. In this volume.
11. Elizabeth A. Kendall. Agent software engineering with role modelling. In this volume.
12. Thomas Malone and Kevin Crowstone. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.
13. Simon Miles, Mike Joy, and Michael Luck. Designing agent-oriented systems by analysing agent interactions. In this volume.
14. James Odell, H. Van Dyke Parunak, and Bernhard Bauer. Representing agent interaction protocols in UML. In this volume.
15. Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies and Applications.* Springer-Verlag, December 2000.
16. George A. Papadopoulos. *Models and Technologies for the Coordination of Internet Agents: A Survey*, chapter 2. In Omicini et al. [15], December 2000.
17. George A. Papadopoulos and Farhad Arbab. Coordination models and languages. *Advances in Computers*, 46:The Engineering of Large Systems:329–400, August 1998.
18. Joav Shoham and Moshe Tennenholtz. Social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73, 1995.
19. Munindar P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):55–61, December 1998.
20. Peter Wegner. Why interaction is more powerful than computing. *Communications of the ACM*, 40(5):80–91, May 1997.
21. Michael J. Wooldridge. Agent-based software engineering. *IEE Proceedings on Software Engineering*, 144(1):26–37, February 1997.
22. Michael J. Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
23. Michael J. Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, September 2000.
24. Franco Zambonelli, Nicholas R. Jennings, Andrea Omicini, and Michael J. Wooldridge. *Agent-Oriented Software Engineering for Internet Applications*, chapter 13. In Omicini et al. [15], December 2000.
25. Franco Zambonelli, Nicholas R. Jennings, and Michael J. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In this volume.