An agent-oriented approach to service analysis and design

Hoa Khanh Dam and Aditya Ghose

School of Computer Science and Software Engineering
University of Wollongong
Northfields Av, Wollongong, NSW 2522, Australia
hoa@uow.edu.au, aditya@uow.edu.au

Abstract. The agent paradigm, with its new way of thinking about software systems as a collection of autonomous, flexible and robust agents, offers a promising solution for modelling and implementing distributed complex systems. Intelligent agents and services share a substantial number of key concepts such as autonomy, reactivity, loose coupling and strong encapsulation. There has been, however, little work on leveraging such a deep connection between agents and services. In this paper, we argue that agent-oriented software engineering (AOSE) provides an important basis for service analysis and design at the business service level. In particular, we show how concepts and techniques in AOSE can be used to analyse and model business services in the context of service ecosystems.

1 Introduction

We now live in a growing services-based economy in which every product today has virtually a service component to it [1]. In this context, many services interact with one another in different ways in order to meet growing customer demands. Business domains involving large and complex collections of loosely coupled services provided by autonomous enterprises are becoming increasingly prevalent. For example, there are multiple services on offer at an international airport. Some rely on others for execution. There is a passenger transport service (taking a person from one airport to another), which relies on the baggage handling service, a security screening service, a business class lounge service and so on. Each of those services can be offered in other business contexts. For instance, the baggage handling service can be independently used for cargo air-freighting, or the security screening service can be independently offered in other high security venues. Such interactions among and between independent and autonomous services are what define a *service ecosystem* [2]. The emergence of such an ecosystem can be seen in various places such as in the form of Shared Service Centres providing central, standardised services from different agencies or departments in the public sector [3] or a Web service ecosystem on the Internet where Web services providers are interconnecting in their offerings in unforseen ways [4].

Therefore, there is an increasing demand to design and build high quality service ecosystems. However, developing such an ecosystem of services is a challenging task due to their loosely coupled nature and openness, and the autonomy of their participants. In fact, each service within an ecosystem are autonomous in which they should operate independently and make decisions without direct intervention of other business partners. Individual services should have their own thread of control and have their own objectives. In order for services within an ecosystem to fulfil both their individual and overall objectives, they need to interact with one another. However, establishing loosely coupled collaboration between autonomous services is still a critical challenge in developing a service ecosystem [5]. In this context, a key question is how to analyse, design and model services, their capabilities and their related business processes in such a way that services have autonomy over their choice of action and have ability to initiate (and respond to) interactions in a flexible manner.

Since the 1990s, intelligent agent technology has evolved rapidly along with a growing number of agent languages, architectures, and theories proposed in the literature. The emerging agent-oriented paradigm, with its potential to significantly improve the development of high-quality and complex systems, has attracted an increasing amount of interest from the research and business communities. Indeed, there have been numerous agent-based applications in a wide variety of domains such as weather alerting [6], business process management [7], holonic manufacturing [8], e-commerce, and information management [9]. A software agent is a piece of software which is situated in an environment, acts on its own and interacts with other similar entities to achieve some design goals [10]. An agent also works pro-actively to pursue certain goals while, at the same time, it responds in a timely fashion to changes that occur in its environment. Agents can interact with other agents and humans with the aim of accomplishing their goals. This agent view provides a well suited level of abstraction for modelling, an effective way of decomposing, and an appropriate method for dealing with the dependencies and interactions in complex software systems [11].

There are strong connections between between agents and services, and are to some degree recognized and addressed in the literature (e.g. [12, 13]). Similarly to agents, services can be can be viewed as autonomous, reactive components in a loosely-coupled architecture. Principles such as strong encapsulation, loosely coupling are well-supported in agent-oriented methodologies. This paper will argue that agent-oriented software engineering provides an important basis for analysis and modeling of service ecosystems. We will show that decomposing the problem space of a service ecosystem in an agent-oriented way is very effective. We will also explain why it is appropriate to apply an agent-oriented philosophy to the modelling and managing relationships and interactions within service ecosystems in such a way that the dependencies and interactions in those complex ecosystems are effectively dealt with.

The paper is structured as follows. In the next section, we provide a brief overview of agent-oriented software engineering and highlight some of its key advantages. We then describe how AOSE can be adapted to service analysis and

design in section 3. Finally, we conclude and outline some future directions for this research in section 4.

2 Agent-Oriented Software Engineering

Agent Oriented Software Engineering (AOSE) is a promising new approach to software engineering that uses the notion of agents as the primary method for analysing, designing and implementing software systems [11]. The effectiveness of AOSE resides in its ability to translate the distinctive features of agents into useful properties of (complex) software systems and to provide an intuitive metaphor that operates at a higher level of abstraction compared to the object oriented model.

The technical embodiment of agency can lead to reduced coupling, resulting in software systems that are more modular, decentralized and changeable. Indeed, the autonomous property of agents can be viewed as encapsulating invocation. Any publicly accessible method of an object can be invoked externally, and once the method is invoked the object performs the corresponding actions. On the other hand, when receiving a message, an agent has control over how it deals with the message. This ability to encapsulate behaviour activation (action choice) is very useful in open environments in which the system consists of organisations that have different goals [11]. Additionally, the robustness, reactiveness and pro-activeness also results in reduced coupling [14]. Once an agent acquires a goal, it commits to achieve the goal by possibly trying different alternatives in responding to changes in the environment. This means that there is no need for continuous supervision and checking since the agent solely takes responsibility for accomplishing its adopted goals. As a result, it leads to less communication and thus reduced coupling.

Loose coupling and strong encapsulation brought by agents are important, especially because they facilitate the process of evolving software by localising changes to an agent or a group of agents. For instance, the BDI architecture (discussed in the previous section) can be used to model and implement goal-directed process selection [15]. Traditionally, the calling process contains the names of the called processes (and possibly other information such as the locations, the data needs, or even the implementation), and the conditions specifying which (process) to call in which circumstance. The major disadvantage of this conventional approach is that the calling process is dependent on the called processes, and thus they are not able to be developed independently of one another. A goal-directed approach can separate the conditions of use from the calling processes and place them in the called processes. As a result, processes become loosely coupled and process selection is made dynamically at run time based on the usage context. In addition, if any chosen process fails, the call is made again (i.e. reposted) and a new matching process is invoked. This offers a better and more automatic handling of exceptions or failures. Furthermore, called processes can be created or changed without affecting the existing ones and the calling process.

These benefits multiply each time the called process is reused in other calling processes.

In addition, agents can lead to the expansion of functionalities, complexities and quality of the real world applications [16]. For example, multi-agent systems with a number of autonomous agents suits the highly distributed environment where such agents are able to act and work independently to each other. In addition, the inherently robust and flexible properties of multiagent systems allow them to work in a more dynamic and/or open environment with error-prone information sources. These properties significantly increase the reliability and failure-tolerance of the system in terms of autonomously recovering from failure and adapting to changes in the environment. Therefore, issues such as improving the unforeseen reuse of software components, developing self-managed software systems can be better addressed using the ideas of multi-agents.

It has also been argued that AOSE, equipped with the rich representation capabilities of agents, is suitable (and reliable) for modelling complex organisational processes [17, 9, 16]. Jenning and Wooldridge in [18, 11] have shown that agent-orientation facilitates complexity management in three aspects: decomposition, abstraction, and organisation. Firstly, they argue that decomposing the problem space of a complex system in an agent-oriented way is very effective. Secondly, they are able to demonstrate the ability of agents in representing high-level abstractions of active entities in a software system, and consequently reducing the gap between business users and system architects. Finally, they explain why it is appropriate to apply an agent-oriented philosophy to the modelling and managing of organisational relationships in such a way that the dependencies and interactions in those complex organisations are effectively dealt with.

As agents have been increasingly recognised as possibly the next prominent paradigm of developing software, there has been a growth of interest in agent-oriented software engineering. A significant amount of AOSE work has focussed on developing new methodologies and tools for software development using the agent concepts. In fact, as far as we are aware of, there have been nearly fifty agent-oriented methodologies proposed to date [19]. Those methodologies (e.g. Tropos [20], Gaia [21], Tropos [20], Prometheus [14], O-MaSE [22], PASSI [23] etc.), Prometheus [14]) offer notations and models, methods and techniques, processes and (for some methodologies) tool support that a software developer can use to develop an agent-based application. Recent studies (e.g. [24]) have shown that AOSE methodologies provide reasonable support for basic agent-oriented concepts such as autonomy, mental attitudes, pro-activeness, and reactiveness. In the next section, we will show how several ideas and techniques proposed in AOSE methodologies can be adapted to be used in the context of service analysis and design.

3 The Case for AOSE to Service Analysis and Design

3.1 Service identification

A crucial task of service analysis and design is identifying service candidates. The importance of service identification is amplified when the target system is a service ecosystem which consists of a large number of services. Existing work (e.g. [25–27]) tend to focus on proposing techniques for describing and decomposing business services and fail to address the important issue of how such services can be identified at the first place. Since services are the fundamental entities in service-based systems, we believe that a critical requirement for an service analysis and design methodology is to assist the developers identify the services constituting the system.

A service ecosystem can be considered as an organisation of services. This view matches with AOSE methodologies. In fact, a large number of AOSE methodologies adopt an organisational view of the world and encourage a designer to think of building agent-based systems as a process of organizational design. The software system organisation is similar to a real world organisation. It has a certain number of entities playing different roles. For instance, a university organisation has several key roles such as administration, teaching, research, students, etc. These roles are played by different people in the university such as managers, lecturers, students, etc. Different roles in an organisation interact with each other to achieve their own goals and also to contribute towards the overall goals of the organisation.

Based on that motivation, a common technique used in most of AOSE methodologies to deal with agent identification is to start from identifying roles. Agents are then formed by grouping these roles into "chunks". There are different techniques and models provided by AOSE methodologies to help the designers group or map these roles into agents. For instance, Prometheus [14] provides clear techniques to deal with agent identification in terms of group functionalities into agents. This is based on considerations of both cohesion and coupling - one wants to reduce coupling and increase cohesion. Putting agents that write or read the same data together seems to reduce coupling between agents. In other words, functionalities that use and/or produce the same data tend to belong to the same agent.

Such techniques are also particulary useful in identifying services. In fact, a service ecosystem can also be viewed as a society or organisation. Hence, in order to identifying the constituting services, a natural way is to identify roles, their goals and their relationships. Roles allow for a combination of both top-down and bottom-up design. They are identified by a top-down process of goal development. At the same time, they provide a bottom-up mechanism for determining service types and their responsibility. Let us take the example of a pizza restaurant. A natural first step is to identify business goals of the pizza restaurant. Examples of such goals include making pizza, delivering pizza, managing inventory, ordering pizza and so on. The next step is to identify various roles participating in the operation of a pizza restaurant by grouping related goals.

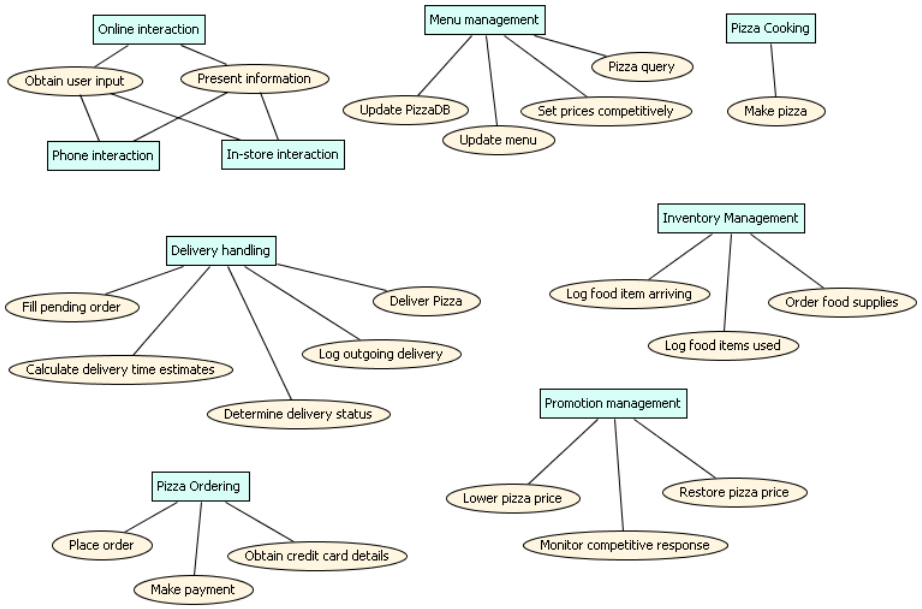

Fig. 1. Roles for the pizza restaurant

For example, there is a role responsible for making pizza, a role for managing the inventory, a role for handling delivery and so on. Figure 1 shows how different roles in a pizza restaurant are identified and represented using Prometheus, a prominent agent-oriented methodology [14]. Such roles provide a foundation for service identification in which services play multiple roles (e.g. pizza order service, pizza transportation service, and pizza cooking service). Roles can also serve as an indication for defining the capabilities which a service offer.

3.2 Service interactions

The current standard approach for designing service interactions is message-centric. Specifically, the design process is driven by messages exchanged during the interaction and tends to focus on data and control flows. In addition, interactions are also defined by interaction protocols which specify the allowable sequence of messages exchanged between service providers and service consumers (e.g. the Unified Service Description Language[1]). Such a message-centric approach however poses several limitations in designing interactions for service ecosystems. Firstly, as we have earlier argued, in a service ecosystem, services should be flexible and robust in terms of pursuing their goals by attempting alternatives in spite of failures. Interaction protocols however restricts the flexibility and robustness of services. Alternative means to achieving the interaction's

[1] `http://www.internet-of-services.com`

objectives is limited to the options the designer have defined. Although it is possible to add an unlimited number of alternatives, a large number of alternatives described using message-centric notations leads to protocols that are difficult to understand and manage. Therefore, in practice message-centric modelling results in brittle service interactions that are neither flexible or robust. Secondly, since the message-centric approaches are based on low-level abstractions, they tend to fail to conceptually capture the business intent of the interactions between services. Existing service interaction protocols tend to over-constraint the business behaviour of participant business services.

The above issues have been recognised in the agent community and recent work in AOSE have proposed a number of alternative approaches design the interactions which are not driven by messages. These approaches shift the focus onto a higher level of abstraction which are more suitable to support complex, dynamic, interactions. In particular, a number of approaches (e.g. [28, 29]) proposes to design agent interactions using social commitments, in which agent participants progress through interactions by making and fulfilling commitments to each other. Other approaches proposes to design interactions on the basis of agent plans and goals. For instance, the Hermes methodology [30] proposes to model interactions in terms of interaction goals, available actions and constraints. Based on the constraints defined by the designer, the agents then work out the allowable message sequences to use during the interactions. In addition, this approach allows the designer to model failure recovery: if a given interaction goal fails then they may specify that accomplishing a previous interaction goal may allow the interaction to continue. For example, if booking a hotel for particular dates cannot be achieved, then rolling back and finding alternative travel dates may solve the problem.

In those alternative approaches, the designer does not need to define legal message sequences explicitly but instead they emerge from the interaction due to the agents' need to fulfil their commitments or achieve their interaction goals. This increases the flexibility and robustness of interactions in terms of providing more valid message sequences than what a designer could have explicitly defined. In addition, those approaches are able to capture the business intent (in forms of goals and commitments) of the interactions.

In our view, those AOSE methodologies can be adapted to design flexible and robust interactions in service ecosystems. Such interaction modelling facilities in AOSE can be useful for service touchpoint modelling which is a critical issue for services modelling. In fact, service interactions take place at *service touchpoints* [31]. Through touchpoints the service is experienced and perceived with all the senses. In a service ecosystem, a provider can deliver services across multiple touchpoints such as the Internet, self-service technologies or face-to-face communication. As an example, many pizza restaurants (e.g. Pizza Hut) now allow their customers to order over the phone, the Internet, or the traditional in-store service desks (refer to figure 1). Touchpoints modeling is critical for services since touchpoints have always been considered as the crucial moment where the consumer judges service quality and the service value revealed. In this context,

flexibility and robustness are key to the success of a touchpoint. In addition, it is the consistency across the many touchpoints that will affect quality perception. In order to create such consistency, the design of service interactions should be driven by interaction goals or commitments to capture the business intents at a higher abstraction level than the traditional message-centric approaches. For example, the passenger can check-in at different touchpoints, i.e. using different check-in facilities, each of which is different in terms of how the actual interaction takes place. However, they all share the same goals, e.g. successful check-in a passenger, and should be the basis for design interactions.

3.3 Business process modelling

One of main challenges facing the service oriented computing community is to develop dynamic and adaptive processes for service composition and orchestration [32]. This requires that a business process should be able to pro-actively adapt itself quickly to respond to environmental demands and changes. We believe that such a requirement should be addressed at the modelling level. With this respect, the *Belief-Desire-Intention* (BDI) model [33], one of the most well-established and widely-used agent models, can offer a solution to model dynamic and adaptive processes. The key concepts in the BDI model are: *beliefs*, i.e. representing information about the environment, the agent itself, or other agents; *desires*, i.e. representing the objectives to be accomplished, i.e. goals; and *intentions*, i.e. representing the currently chosen courses of action to pursue a certain desire that the agent has committed to pursuing, i.e. plans. More specifically, BDI agents have a collection of pre-defined plan recipes (or types), usually referred to as a plan library [34]. Each plan consists of: (a) an invocation condition which defines the event that triggers this plan (i.e. the event that the plan is *relevant* for); (b) a context condition (usually referring to the agent's beliefs) which defines the situation in which the plan is *applicable*, i.e. it is sensible to use the plan in a particular situation; and a plan body containing a sequence of primitive actions and subgoals that are performed for plan execution to be successful. It should be noted that subgoals can trigger further plans.

A typical execution cycle that implements the decision-making of an agent can be viewed as consisting of the following steps. First, an event is received from the environment, or is generated internally by belief changes or plan execution. The agent responds to this event by selecting from its plan library a set of plans that are relevant (i.e. match the invocation condition) for handling the event (by looking at the plans' definition). The agent then determines the subset of the relevant plans that is applicable in terms of handling the particular event. The determination of a plan's applicability involves checking whether the plan's context condition holds in the current situation. The agent selects one of the applicable plans and executes it by performing its actions and sub-goals. A plan can be successfully executed, in which case the (sub-)goal is regarded to have been accomplished. Execution of a plan, however, can fail in some situations, e.g. a sub-goal may have no applicable plans, or an action can fail, or a test can be false. In these cases, if the agent is attempting to achieve a goal, a mechanism

that handles failure is used. Typically, the agent tries an alternative applicable plan for responding to the triggering event of the failed plan. It is also noted that failures propagate upwards through the event-plan tree: if a plan fails its parent event is re-posted; if this fails then the parent of the event fails and so on.

BDI agents offer two important qualities: robustness and flexibility. BDI agents are robust since they are able to pursue persistent goals over time (i.e. pro-activeness). In other words, agents will keep on trying to achieve a goal despite previously failed attempts. In order to be able to recover from such failures, agents have multiple ways of dealing with a given goal and such alternatives can be used in case any of them fail. This gives agents flexibility in terms of exercising choice over their actions. Flexibility and robustness are considered as useful qualities that a software system should possess, especially if it operates in complex, dynamic, open and failure-prone environments.

Those properties of the BDI model also offer a suitable solution for the business process modelling within service design. More specifically, the BDI architecture can be used to model and implement goal-directed process selection [15]. Traditionally, the calling process contains the names of the called processes (and possibly other information such as the locations, the data needs, or even the implementation), and the conditions specifying which (process) to call in which circumstance. The major disadvantage of this conventional approach is that the calling process is dependent on the called processes, and thus they are not able to be developed independently of one another. A goal-directed approach can separate the conditions of use from the calling processes and place them in the called processes. Such conditions form different possible contexts of the process. As a result, processes become loosely coupled and process selection is made dynamically at run time based on the usage context. In addition, if any chosen process fails, the call is made again (i.e. reposted) and a new matching process is invoked. This offers a better and more automatic handling of exceptions or failures. Furthermore, called processes can be created or changed without affecting the existing ones and the calling process. These benefits multiply each time the called process is reused in other calling processes.

3.4 Value modelling

A key component of services modelling is value modelling. It is important that we should understand how a service delivers value to its stakeholders. Ideally, value models should be stakeholder-specific - in other words, we need to be able to account for the fact that a service delivers different kinds of value to distinct stakeholders, and in different ways. A value model is critical in service design - it provides a guiding framework that ensures that a service design maximizes the value it delivers to its key stakeholders. Value models, when correlated to service design components, also support service re-design. When a service must be modified to account for changes in the operating context, for instance, a value model can help decide which components of a service design may be discarded (when there are options, we pick those components that deliver lower value) and

which components might be modified. Value models can represent value on a variety of scales, both quantitative and qualitative, or via preference orderings over design elements.

A key insight that is often ignored in the literature on services science is the fact that a value model is fundamentally a requirements model. A service delivers value by providing certain functionalities, under certain non-functional or quality-of-service constraints. Goal models, which are central to most AOSE frameworks, provide an appropriate basis for modelling the functional aspects of service value. Softgoal models, also common in AOSE frameworks (e.g. [20]), enable us to model the non-functional aspects of service value.

4 Conclusions and Future Work

Together with the grow of services across different sectors in the society, service ecosystems emerge as a complex collection of services in which they interact with one another to meet customer demands. In these ecosystems, participant services are highly independent, autonomous, flexible and reactive to environment changes. Due to those complexities, designing high quality service ecosystems is a very challenging task.

However, such challenges have previously been addressed to a certain extend by the agent paradigm, another important technology which has emerged since the past decade. In this paper we have argued that agent-oriented software engineering (AOSE) methodologies provide a number of techniques that can be adopted to the analysis and design of services. More specifically, we have shown that the organisational view of a system which AOSE methodologies take is also suitable to service ecosystems. In addition, we have described a number of emerging approaches in AOSE which shift the focus of interaction design from messages to goals and commitments. Such approaches are suitable for designing service interactions in service ecosystems since they allow the designer to work at a higher level abstraction and to define flexible and robust interactions. We have also briefly described the BDI model, which is widely used by AOSE methodologies to model agent plans, and have shown that this model is suitable to model the business processes of services.

Those key ideas proposed in this paper can to a number of directions for further research. In particularly, we plan to develop a methodology for service analysis and design which systematically adopts AOSE techniques that are suitable for services as discussed in this paper. This methodology would specifically support for modelling service ecosystems in several key areas including service identification, service interactions, business processes, and value modelling. Future work is also needed to investigate how ideas and techniques proposed by AOSE methodologies to support the design of negotiation, cooperation, and teamwork can be used for modelling service ecosystems.

References

1. Paulson, L.D.: Services science: A new field for today's economy. Computer **39**(8) (2006) 18–21
2. Sawatani, Y.: Research in service ecosystems. In: Proceedings of Management of Engineering and Technology (PICMET'07), Portland, USA (2007) 2763–2768
3. Janssen, M., Wagenaar, R.: An analysis of a shared services centre in e-government. In: HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 5, Washington, DC, USA, IEEE Computer Society (2004) 50124.2
4. Barros, A.P., Dumas, M.: The rise of web service ecosystems. IT Professional **8**(5) (2006) 31–37
5. Ruokolainen, T., Kutvonen, L.: Managing interoperability knowledge in open service ecosystems. In: Proceedings of the 13th Enterprise Distributed Object Computing Conference Workshops, EDOCW, Auckland, New Zealand, IEEE (September 2009) 203–211
6. Mathieson, I., Dance, S., Padgham, L., Gorman, M., Winikoff, M.: An open meteorological alerting system: Issues and solutions. In Estivill-Castro, V., ed.: Proceedings of the 27th Australasian Computer Science Conference, Dunedin, New Zealand (2004) 351–358
7. Burmeister, B., Arnold, M., Copaciu, F., Rimassa, G.: BDI-Agents for agile goal-oriented business processes. In Padgham, Parkes, Müller, Parsons, eds.: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal (May 2008) 37–44
8. Monostori, L., Váncza, J., Kumara, S.: Agent based systems for manufacturing. CIRP Annals-Manufacturing Technology **55**(2) (2006) 697–720
9. Munroe, S., Miller, T., Belecheanu, R.A., Pěchouček, M., McBurney, P., Luck, M.: Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. Knowledge Engineering Review **21**(4) (2006) 345–392
10. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review **10**(2) (1995) 115–152
11. Jennings, N.R.: An agent-based approach for building complex software systems. Communications of the ACM **44**(4) (2001) 35–41
12. Ghose, A.: Industry traction for MAS technology: would a rose by any other name smell as sweet. Int. J. Agent-Oriented Softw. Eng. **3**(4) (2009) 397–401
13. Cabri, G., Leonardi, L., Puviani, M.: Service-oriented agent methodologies. In: WETICE '07: Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Washington, DC, USA, IEEE Computer Society (2007) 24–29
14. Padgham, L., Winikoff, M.: Developing intelligent agent systems: A practical guide. John Wiley & Sons, Chichester (2004) ISBN 0-470-86120-7.
15. Georgeff, M.: Service orchestration: The next big challenge. DM Direct Special Report (June 2006)
16. Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink (2005)
17. Jennings, N.R., Sycara, K., Wooldridge, M.: A roadmap of agent research and development. Journal of Autonomous Agents and Multi-Agent Systems **1**(1) (1998) 7–38

18. Jennings, N.R., Wooldridge, M.: Agent-Oriented Software Engineering. In Garijo, F.J., Boman, M., eds.: Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99). Volume 1647., Springer-Verlag: Heidelberg, Germany (30– 2 1999) 1–7
19. Henderson-Sellers, B., Giorgini, P., eds.: Agent-Oriented Methodologies. Idea Group Publishing (2005)
20. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3) (2004) 203–236
21. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Transactions on Software Engineering and Methodology **12**(3) (2003) 317–370
22. DeLoach, S.A.: Engineering organization-based multiagent systems. In Garcia, A.F., Choren, R., de Lucena, C.J.P., Giorgini, P., Holvoet, T., Romanovsky, A.B., eds.: Software Engineering for Multi-Agent Systems IV, Research Issues and Practical Applications. Volume 3914 of Lecture Notes in Computer Science., Springer (2005) 109–125
23. Cossentino, M.: From requirements to code with the PASSI methodology. In B., H.S., Giorgini, P., eds.: Agent-Oriented Methodologies. Idea Group Inc. (2005) 79–106
24. Dam, K.H., Winikoff, M.: Comparing agent-oriented methodologies. In Giorgini, P., Henderson-Sellers, B., Winikoff, M., eds.: Agent-Oriented Information Systems. Volume 3030 of Lecture Notes in Computer Science., Springer (2003) 78–93
25. Scheithauer, G., Augustin, S., Wirtz, G.: Describing services for service ecosystems. (2009) 242–255
26. Dhanesha, K.A., Hartman, A., Jain, A.N.: A model for designing generic services. In: SCC '09: Proceedings of the 2009 IEEE International Conference on Services Computing, Washington, DC, USA, IEEE Computer Society (2009) 435–442
27. Lê, L.S., Ghose, A., Morrison, E.: Definition of a description language for business service decomposition. In: Proceedings of First International Conference on Exploring Services Sciences (IESS 1.0), Geneva, Switzerland (2010)
28. Winikoff, M.: Designing commitment-based agent interactions. In: IAT '06: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology, Washington, DC, USA, IEEE Computer Society (2006) 363–370
29. Yolum, P., Singh, M.P.: Flexible protocol specification and execution: applying event calculus planning using commitments. In: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM (2002) 527–534
30. Cheong, C., Winikoff, M.: Hermes: a methodology for goal oriented agent interactions. In: AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, New York, NY, USA, ACM (2005) 1121–1122
31. Bitner, M.: Evaluating service encounters: the effects of physical surroundings and employee responses. Journal of Marketing **54**(2) (1990) 69–82
32. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. Computer **40**(11) (2007) 38–45
33. Bratman, M.E.: Intentions, Plans, and Practical Reason. Harvard University Press, Cambridge, MA (1987)
34. Rao, A.S., Georgeff, M.P.: BDI-agents: from theory to practice. In: Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco (1995)