# AgentPrIMe: Adapting MAS Designs to Build Confidence

Simon Miles[1], Paul Groth[2], Steve Munroe[2], Michael Luck[1], Luc Moreau[2]

[1] Department of Computer Science, Kings College London, UK
[2] School of Electronics and Computer Science, University of Southampton, UK

**Abstract.** The products of systems cannot always be judged at face value: the process by which they were obtained is also important. For instance, the rigour of a scientific experiment, the ethics with which an item was manufactured and the use of services with particular licensing all affect how the results of those processes are valued. However, in systems of autonomous agents, and particularly those with multiple independent contributory organisations, the ability of agents to choose how their goals or responsibilities are achieved can hide such *process qualities* from users. The issue of ensuring that users are able to check these process qualities is a software engineering one: the developer must decide to ensure that adequate data is recorded regarding processes and safeguards implemented to ensure accuracy. In this paper, we describe AgentPrIMe, an adjunct to existing agent-oriented methodologies that allows system designs to be adapted to give users confidence in the results they produce. It does this by adaptations to the design for *documentation, corroboration, independent storage* and *accountability.*

## 1 Introduction

Agent-based systems have particular qualities that require their activity to be justified to their users. First, since they are based on autonomous components, decisions that make use of expert knowledge or have significant consequences can be handled by software, and so the decisions made by such software must be seen to be reliable if the software is to be widely adopted. In addition, by having multiple, distributed points of control, an application may rely on services not under the authority of the user, and whose side-effects may not be apparent to the user: a user may wish to know that the services do not produce their results in an undesirable way, such as being illegal, unethical, etc. Finally, in systems where agents represent localised concerns of distributed users, it is important to know that agents have not released private information more widely than desired.

At some level, this problem has been well researched. There are already approaches to formally specify a multi-agent system, enabling developers to verify its desirable properties [9]. However, this does not in itself inform developers about what factors need to be considered, nor is it (commercially) realistic to assume fine-grained knowledge of third-party services used in an application.

Mechanisms have been designed to guide agent behaviour towards reliable results or to constrain agent behaviour to only desirable results, including contracts, norms, protocols, trust evaluations etc.

Nevertheless, we argue that, even with this breadth of beneficial technology, there are significant outstanding issues. First, agent-based systems must be designed not just to be reliable but to make their reliability apparent to users if they are to have *confidence* in the system. Second, the above mechanisms concentrate on the value or otherwise of *results* or the *cost* of achieving those results, both aspects of the system that can be immediately judged by the user or an agent acting on their behalf. Because of this emphasis, other, hidden but still important, aspects are ignored. In particular, the mechanisms do not address how to determine *process qualities* that are not immediately apparent in the result returned by an agent but have an impact on its worth. Examples of important process qualities occur in many domains, such as the following.

- The rigour of the scientific experiment that produced some result.
- The ethics (fair trade, environmental impact, etc.) of the process that led to the sale of an item.
- The use of services with licenses that make a result unpatentable.
- The actual inter-dependence of two apparently independent recommendations.

The qualities of the process that led to a result are all evident in the *provenance* of that result, i.e. everything that caused the result to be as it is. For the provenance of a result, and process qualities evident from it, to be made apparent to a user requires that an agent-based system be engineered to *record* adequate information to determine both *(1)* what has occurred in the system prior to the result being produced, and *(2)* which of those events are causally related to the eventual result.

However, in a system of flexible autonomous agents, such agents may lie or collude to hide the actions they have taken where it is in their interests to do so (as is true in the four process examples above). Similarly, without specifically designing a system that prevents agents' inaccuracy, a user can be misled. Therefore, we argue that agent-oriented designs must be specifically adapted to mitigate for inaccuracy and provide confidence that users can determine exactly how a multi-agent system came to produce a result.

In this paper, we describe *AgentPrIMe*, an additional stage for existing methodologies. It is used, firstly, for determining what information needs recording and how to adapt the relevant agents to do so. Then, it tackles what must be established of an agent owned by a third-party in order to rely on it to provide compatible and verifiable information regarding provenance.

## 2 AgentPrIMe

A *methodology fragment* [8] is a software engineering procedure that is used in addition to the usual stages of a methodology when designing an application. It aims to add or ensure some functionality of the system, that may otherwise not be guaranteed by the original methodology. Aspect-oriented software engineering [6] provides an example of methodology adjuncts that provide functionality pervading across a design (usually object-oriented). Others have applied aspect-orientation to agent-based systems [3], but we do not use the aspect concept here because, while it is not entirely inappropriate, it carries connotations of cutting across agents in a way that pre-supposes that the process they are involved in is fixed at design-time. Process qualities are concerned with processes that have already occurred in a system that may be flexible, open and unreliable.

A desirable quality of a methodology adjunct is *methodology-neutrality*, meaning that it is general and sufficiently well-defined to be applied as part of as many methodologies as possible. This is a distinct quality from the comparable requirement of methodologies (and their adjuncts) of being *widely applicable* to a range of applications.

AgentPrIMe is a methodology adjunct for agent-oriented software engineering methodologies. We will refer to the methodology to which it is acting as an adjunct as the *extended methodology.* The outcome of applying AgentPrIMe is a set of *adaptations* to be applied to a system design, so that queries regarding provenance can be reliably answered. It builds on an existing methodology adjunct, Provenance Incorporation Methodology (PrIMe), described elsewhere [11], which is concerned with adapting software to help users determine provenance of results, but considers only service-oriented systems. In particular, PrIMe does not address issues relevant to an agent-oriented design, where autonomous components choose their own methods to achieve their goals and so may be dishonest.

There are two aims of AgentPrIMe: *(1)* to make the provenance of results available to users of the system, and *(2)* to ensure that, as far as possible, the provenance is accurate even when agents in the system may be unreliable. Specifically, AgentPrIMe has two phases, described in detail in the following sections.

– Identify the causes of agent actions in the design, instances of which are recorded as the agents act. This phase results in adaptations to agents so that they record such causes for users to later query.
– Identify where additional guarantees of accuracy are required, so as to be able to rely on what agents have recorded. This phase results in adaptations to the interactions between agents, so that users can have more confidence that what agents have recorded is accurate.

AgentPrIMe relies on understanding the types of agents that will exist within a system, so that their effects in processes and the interactions possible between those agents can be understood. It can affect both how those agents are ultimately implemented, and may alter the possible interactions between them, as will be seen in the subsequent sections. These dependencies mean that AgentPrIMe is ideally applied at a particular point in the extended methodology, when the design is sufficiently well developed to adapt but not so far developed that effort is wasted. To be more concrete, we specify below at which point AgentPrIMe would apply when using various methodologies that the reader may be familiar with.

- In Gaia [14], AgentPrIMe must be applied after the *agent model* and *acquaintance model* have been completed. This is because it applies to *agent types*, where the functionality of an agent of each type is well-defined, and the interactions between them dictated by the acquaintance model.
- In MaSE [2], AgentPrIMe must be applied after the *agent classes* and *conversations* have been created, for analogous reasons to those given for Gaia.
- In Prometheus [13], AgentPrIMe operates on the *agent overview*, after the architecture design and before the detailed design.
- In SODA [12], AgentPrIMe requires the data from the *interaction* and *agent models*, so applies after SODA has completed.

## 3 Causality in Multi-Agent Systems

In this section, we describe the first phase of AgentPrIMe, where system designs are adapted to document the causal relationships between agent actions. This gives users the facilities to determine the provenance of agent actions and outputs. We consider the unreliability of agents in the next section.

### 3.1 Causality within Agents

A key part of AgentPrIMe is to allow agents to document the *causes* of their actions, so that this information can later be used to determine what occurred in a process. The possible causes in a model depend on the extended methodology, but we discuss some examples in this section and then show how these can be generalised in a well-defined way for methodology-neutrality.

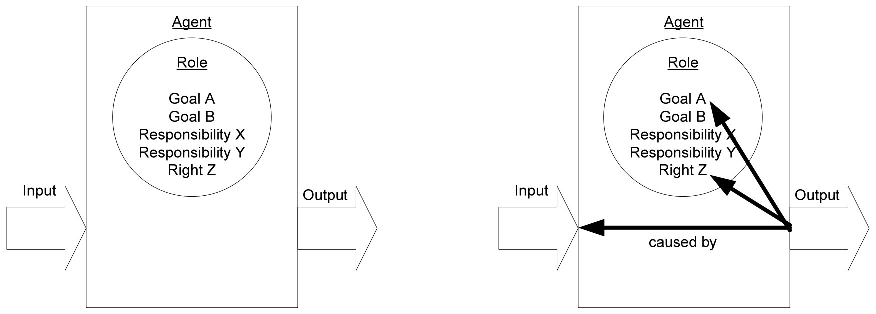

**Fig. 1.** Potential causes of an agent's actions (left) can be made explicit and recorded (right)

A variety of factors influence an agent's behaviour at a given instant, as illustrated in the examples summarised in Figure 1 (left). Here, we are concerned with behaviour that affects the environment, i.e. actions, shown as the *output* of the agent. Depending on the agent model used by the extended methodology, the influencing factors can include the agent's *goals*, *responsibilities* or *rights*. Often, the latter factors are due to the *roles* that the agent is playing within a system at that instant, with the goals and responsibilities having been allocated to the roles in applying the extended methodology [7]. Additionally, triggers from the environment, which include messages from other agents, shown as *input* in the figure, influence how an agent acts.

AgentPrIMe, and its supporting technologies, allow an agent to assert the *causal relationship* between two *occurrences*. These assertions, called *relationship p-assertions*, can be stored for later interrogation by a user, as discussed further below. Applied to an agent design, this means that relationships can be asserted between an output (the effect) and the inputs, goals, responsibilities, and so on that caused it to take place. These relationships are depicted in Figure 1 (right). Using the examples already described, these causal relationships can be used to assert:

- that an output message was sent in response to an input message;
- that an action was taken to attempt to fulfil a goal;
- that an action was taken because it was part of the reponsibilities of the agent; or
- that an action was taken because it was allowed for by a right of the agent.

In recent work [10], we have discussed one particular example of this: how the documentation of the causal effects of goals can be used to make applications more robust.

However, the concepts described above are only a subset of those used in agent-oriented methodologies. Others include motivations, beliefs, intentions, adherence to protocols and so on, and many of these may be asserted as causes of an agent's action. In order for AgentPrIMe to be methodology-neutral, we need a general definition of whether something specified as part of applying a methodology is a causal relationship, and adopt the following definition derived from work in the philoshopy of mind [5].

> $E$ was caused by $C$, if $E$ would not have occurred without $C$ not having occurred, all else being equal.

By applying this definition, we can determine whether a particular factor influenced an action regardless of the methodology extended. For example, we can say that a particular action would not have been taken if the agent didn't have a responsibility to do so, or that an action would not have occurred (because it could not) if the agent did not have the right to do so. The important quality of this definition is that it is *system independent*, relying only on a notion of occurrence.

### 3.2 Causality between Agents

One of the causes of an agent's actions discussed above is a message received from another agent. This is of particular interest when examining process qualities: it is not the actions of a single agent that matter but of a set of agents that ultimately produce some result. Therefore, in addition to asserting causal relationships, AgentPrIMe allows agents to assert the inputs it receives and outputs it sends to other agents. These assertions are called *interaction p-assertions*, and, along with relationship p-asssertions, connect together the actions of one agent to those of another.

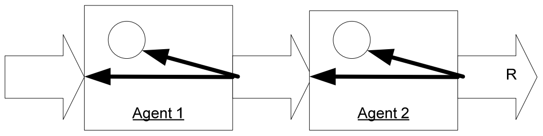

**Fig. 2.** A chain of interactions and causal relationships between agents

A chain of two agents is shown in Figure 2. In this figure, Agent 1 sends a message to Agent 2. This behaviour by Agent 1 is caused by the factors discussed in the previous section, possibly including communications from other agents. Agent 2 may act on the basis of receiving the message, possibly sending messages to other agents. Thus, an adequate collection of interaction and relationship p-assertions provides a connected trail of the process that led to a result. From the result, R, shown in the figure we can follow the causal relationships and interactions back to determine all the factors that ultimately caused it to be as it is. Note that here, we are describing the actual interactions that an agent engages in at run time. How to design agent interactions to best meet system requirements has been addressed by others [1].

### 3.3 The Wrapper Adaptation

The p-assertions described above must be recorded in repositories so that users can later query them. We call such repositories *provenance stores*. Such recording of interaction and relationship p-assertions can be realised in a system by applying a *wrapper* to each agent that is doing the recording, as shown in Figure 3. As messages come into or leave an agent, the wrapper records interaction p-assertions regarding their content, and relationship p-assertions regarding their causes.

### 3.4 Provenance

An important part of our approach is to use a common, open data model for p-assertions. This means that all agents can independently and autonomously

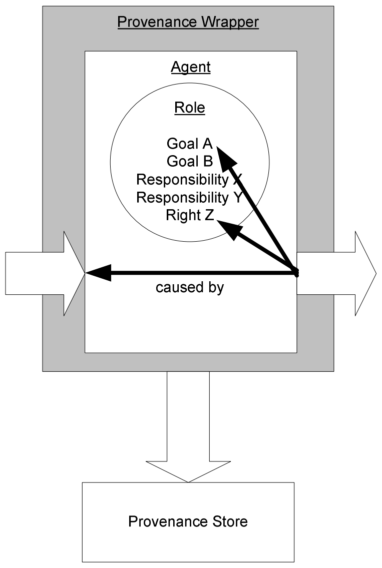

**Fig. 3.** Wrappers are adaptations to agents that automatically document incoming and outgoing messages, and causal relationships, and send them to a provenance store

record documentation of their activities in the same format, and a user can examine and interpret this documentation without relying on implementation details of those agents. The full data model is documented elsewhere [4].

By examining the provenance of a result, we can therefore determine the procedure that was followed to produce it. In theory, this would allow us to check such process qualities as the rigour of a scientific result, or whether businesses with dubious ethical records were used in manufacturing a good. However, doing so depends on the agents involved in a process accurately documenting what they do, an unreasonable assumption in many domains. In the following section, we show how AgentPrIMe tackles the problem of potentially dishonest agents.

## 4 Designing for Accuracy

In this section, we discuss the ways in which agents' inaccuracy can obscure process qualities, and how AgentPrIMe mitigates these problems through *third-party storage*, *accountability* and *corrobration*. It should be emphasised that these solutions do not *guarantee* accurate, honest documentation, but merely reduce the possibilities for deception.

### 4.1 Design Levels

Mitigating for inaccuracy can be expensive, and not every application of Agent-PrIMe needs to incur all of this expense. For instance, a multi-agent system may be completely trusted not to maliciously produce incorrect assertions, e.g. if all agents are owned by a single trusted organisation, but still may do so through error. It is important, therefore, that AgentPrIMe allows developers to apply the degree of mitigation they consider most appropriate for a given application.

We classify types of application, and the design requirements due to them, into three levels, increasing in development cost. A *reliable system* is one in which the agents are assumed always to record complete and accurate documentation, or at least sufficiently complete and accurate that any mitigation would be more costly than it is worth. A *transparent system* is one in which the agents cannot always be trusted to assert correct information but for which there exist ways to corroborate what they have asserted. An *exploitable system* is one in which some agents are free to withold information about their activities or give false information without being detectable. The latter two types of system will be characterised more concretely in the following section.

It is important to note that the systems that need to be adapted to mitigate inaccuracy are exactly those systems that users may suspect of recording inaccurate documentation. The incentive for the designers of such systems to apply the adaptations is that users can check whether they have been applied and will trust the results produced by such systems on that basis. That is, regardless of whether a system is reliable or not, a user can choose to trust results from that system only if it is both *(i)* clear from a result's provenance that it was produced in a legitimate way, and *(ii)* clear from the provenance and other system components described below that the designs were adequate to prevent inaccuracy. AgentPrIMe, therefore, provides benefits to two parties:

– for the user, it provides a way to check that adequate safeguards were in place to ensure the provenance is reliable; and
– for the system designer, it provides a way to give the necessary guarantees of accuracy to a user.

## 4.2 Corroboration

We now characterise the difference between transparent and exploitable unreliable systems, and show how AgentPrIMe requires more adaptations to be applied to the latter.

Returning to the causal chain shown in Figure 2, we note that for every message in the system, two agents are involved: the *sender* and the *receiver*. If both agents record interaction p-assertions documenting the fact and content of the message they sent/received, then one agent's assertion can be used to verify the correctness of the other's assertion. We say that each agent's *view* of the interaction provides *corroboration* of the other view. Therefore, where an interaction involves one reliable agent and one unreliable agent, the latter's view of what occurred can be checked. Note, that this cannot apply to the internal causal relationships: only an agent knows whether its actions were caused by a particular goal, responsibility etc. We argue that the actions that are taken in a system will tend to be more important than the intent behind them for the end user, so that the lack of ability to corroborate internal information is not critical.

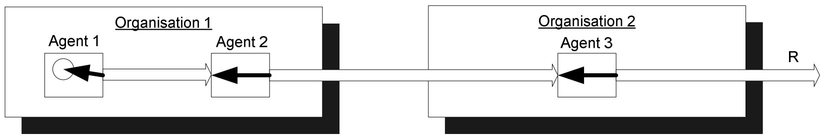

**Fig. 4.** Multiple organisations involved in a process

The kinds of process that cause most problems are those that involve multiple *organisations*, where each organisation owns a set of agents involved in the process. This is problematic because one organisation can provide an honest facade for another; for example, an apparently reliable shop may use an unethical supplier. We depict such a scenario in Figure 4, in which Agent 3 in Organisation 2 produces result R partly on the basis of the operations of Agents 1 and 2 in Organisation 1.

Organisations provide a unit of trust: agents can be grouped into organisations such that all agents in an organisation are trusted independently from those in any other organisation. If, in the process shown in Figure 4, Organisation 1 is trusted, then the system as a whole can be said to be transparent. This is because every agent is either trusted or, if not, every interation they have in a process is with a trusted agent and can therefore be corroborated by examining the p-assertions of the trusted agent.

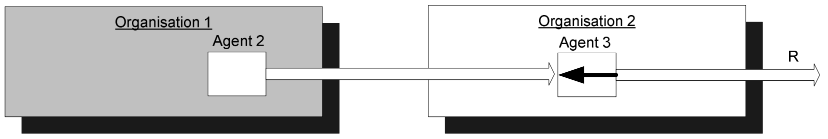

**Fig. 5.** Opaque organisations involve actions that cannot be verified

An alternative situation arises when Organisation 2 is trusted, but Organisation 1 is not. In this case, one of the agents' assertions cannot be corroborated. The situation, from a user's point of view, is shown in Figure 5: only Agents 2 and 3 produce p-assertions that can be relied on. In this case, we say that Organisation 1 is *opaque* because part of its process, possibly a significant part from the user's perspective, is not reliably documented.

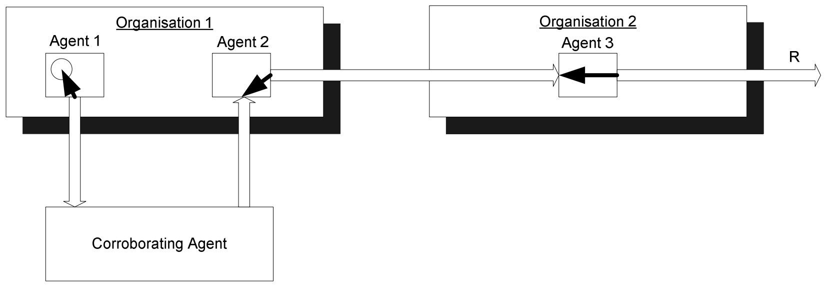

**Fig. 6.** Corroborating agents can be introduced to ensure a process remains transparent

## 4.3 The Corroboration Adaptation

Applying AgentPrIMe to the case above causes additional agent interactions to be introduced to the application process. As shown in Figure 6, the *corroborating agent* is introduced into the process, so that instead of a direct interaction between Agents 1 and 2, this corroborating agent acts as a redirecting intermediary. The corroborating agent must record its own documentation of its interactions, and be trusted by a user to be of value. The agent may be part of an existing trusted organisation, such as Organisation 2, or may in a new organisation created by the user.

## 4.4 The Third-Party Storage Adaptation

The second technique to mitigate inaccuracy is for agents to store documentation in third-party provenance stores, trusted by both the system owner and the user. These repositories should ensure immutability and longevity of the assertions they contain. Therefore, users have assurance that if accurate data is recorded, it cannot later be altered or deleted. Provenance stores independent from the agents recording documentation is recommended for all types of system, even reliable ones.

## 4.5 The Accountability Adaptation

The third technique is to ensure that it is possible to verify the origin of every p-assertion recorded, i.e. which agent created it. This is important for every type of unreliable system, including those that are unreliable through error rather than malice, as it allows the faulty agents to be pinpointed within a system. Accountability can be achieved by each agent applying a digital signature to each p-assertion, with users able to validate a p-assertion's signature when they retrieve it from a provenance store (the store may also do its own checks).

This guards against a particular type of deception that applies to both transparent and exploitable systems: an agent may assert something false but attempt to make it appear that the assertion comes from another, trusted, agent. Without accountability, agents are free to give a completely false view of a process without detection.

## 5 Applying AgentPrIMe

Applying the AgentPrIMe methodology fragment in the context of an agent-oriented methodology requires that the developer knows both at which point to apply it and what steps to take in doing so. With regard to the former point, we have already said that AgentPrIMe is ideally applied at a particular point in the extended methodology, when the design is sufficiently well developed to adapt, but not so far developed that effort is wasted. In practice, given the adaptations described above, this means the point at which (types of) agents have been defined well enough to know the (types of) interactions they will take part in and the causal chains their actions lead to. Depending on the methodology, some adaptations may be best applied even later, when an agents internal structure is defined.

Once a reasonable point in the methodology has been determined, the developer should consider each agent in turn and determine how to wrap the agent to record p-assertions about its activity in a provenance store, preferably a third party one. The form that such wrapping takes depends on technology: the aim is for the agents logic to trigger the recording of p-assertions and anything that achieves this aim is considered to be an instantiation of a wrapper adaptation. Consideration of ensuring accuracy can then begin. First, where possible, an agent should be adapted to sign its p-assertions. Second, each interaction between agents should be considered and, where no agent would be able to corroborate the contents of the interaction, a third party should be added and interactions redirected through it. The choice of third party is based on the developers best guess as to what will be trusted by those others the system is likely to interact with.

In terms of tool support, if an agents internal operations are made explicit, for example as an architecture with plans, then it may be possible to automate the modelling of causation in that agent.

## 6 Conclusions

AgentPrIMe is an extension applicable to existing agent-oriented methodologies that gives users confidence in the results produced by designed systems. Developers applying AgentPrIMe to a design must determine how that design needs to be adapted, firstly to record adequate documentation that exposes the qualities of the process that produced some output of the system, and then to ensure that the documentation itself is reliable through corroboration, independent storage and accountability of agents.

The approach aims to be as *methodology-neutral* as possible, being applicable regardless of the agent-oriented concepts that have been used in designing a system. It does this by relying only on the agents and their interactions, that are present in any multi-agent system, and then defining the *causal relationships*, which define the processes they are involved in, in a system-independent way.

Four design adaptations are defined in this paper:

**Wrapper Adaptation** Adapting agents (or agents of a given type) to record documentation on what they have done and why.

**Corroboration Adaptation** Adapting agent interactions that may be seen as collusion so that an intermediary can provide collaborating evidence of the communications.

**Third-Party Storage Adaptation** Providing storage of documentation that is trusted by both recording agents and users.

**Accountability Adaptation** Adapting agents to sign data before recording it for users to query.

In future work, we will investigate further uses of process documentation recorded by multi-agent systems. For instance, it may be possible to determine whether agents have fulfilled their responsibilities and do not prevent other agents exercising their rights, by examining the documentation recorded. Additionally, we will investigate how the assurances provided by our adaptations can be integrated with the quantitative trust models prevalent in other agent-based research to give an informative measure of reliability to users.

## 7 Acknowledgements

This work was supported in part by the Southampton-Chicago Activity (SOCA) project (EPSRC reference EP/C528131/1). It was also supported in part by the CONTRACT project, which is co-funded by the European Commission under the 6th Framework Programme for RTD with project number FP6-034418. Notwithstanding this fact, this paper and its content reflects only the authors' views. The European Commission is not responsible for its contents, nor liable for the possible effects of any use of the information contained therein.

## References

1. Christopher Cheong and Michael Winikoff. Hermes: Designing goal-oriented agent interactions. In *Agent-Oriented Software Engineering VI: 6th International Workshop (AOSE 2005)*, pages 16–27, Utrecht, Netherlands, 2005.
2. Scott A. DeLoach, Mark F. Wood, and Clint H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.
3. Alessandro Garcia, Uira Kulesza, Claudio Sant'Anna, Christina Chavez, and Carlos J. P. de Lucena. Aspects in agent-oriented software engineering: Lessons learned. In *Agent-Oriented Software Engineering VI: 6th International Workshop (AOSE 2005)*, pages 231–247, Utrecht, Netherlands, 2005.

4. Paul Groth, Sheng Jiang, Simon Miles, Steve Munroe, Victor Tan, Sofia Tsasakou, and Luc Moreau. An architecture for provenance systems. Technical report, Electronics and Computer Science, University of Southampton, October 2006. Available at http://eprints.ecs.soton.ac.uk/12023/.
5. S. Guttenplan. *Introduction to Philosophy of Mind*, chapter An Essay on Mind. Oxford University Press, 1994.
6. Ivar Jacobson and Pan-Wei Ng. *Aspect-Oriented Software Development with Use Cases*. Addison Wesley, 2004.
7. Ivan J. Jureta, Stephane Faulkner, and Pierre-Yves Schobbens. Allocating goals to agent roles during mas requirements engineering. In *Proceedings of the 7th International Workshops on Agent-Oriented Software Engineering‘ (AOSE 2006)*, 2006. Forthcoming.
8. Kuldeep Kumar and Richard J. Welke. Methodology engineering: a proposal for situation-specific methodology construction. In *Challenges and strategies for research in systems development*, pages 257–269, New York, NY, USA, 1992. John Wiley & Sons, Inc.
9. Michael Luck and Mark d'Inverno. Engagement and cooperation in motivated agent modelling. In C. Zhang and D.Lukose, editors, *Distributed Artificial Intelligence Architecture and Modelling: Proceedings of the First Australian Workshop on Distributed Artificial Intelligence*, volume 1087 of *Lecture Notes in Computer Science*, pages 70–84. Springer-Verlag, 1996.
10. Simon Miles, Steve Munroe, Michael Luck, and Luc Moreau. Modelling the provenance of data in autonomous systems. In *Proceedings of Autonomous Agents and Multi-Agent Systems 2007*, page 8 pages, Honolulu, Hawai'i, May 2007.
11. Steve Munroe, Simon Miles, Luc Moreau, and Javier Valquez-Salceda. Prime: A software engineering methodology for developing provenance-aware applications. In *Proceedings of the Software Engineering and Middleware Workshop (SEM 2006)*. ACM Digital, 2006. To appear.
12. Andrea Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In *Proceedings of the First International Workshop on Agent-Oriented Software Engineering (AOSE 2000)*, pages 185–193, 2000.
13. Lin Padgham and Michael Winkoff. Prometheus:a methodology for developing intelligent agents. In *Agent-Oriented Software Engineering III: Third International Workshop (AOSE 2002)*, pages 174–185, Bologna, Italy, July 2002.
14. Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Automous Agents and Multi-Agent Systems*, 3:285–312, 2000.