Engaging Stakeholders with Agent-Oriented Requirements Modelling

Tim Miller¹, Sonja Pedell², Leon Sterling¹, and Bin Lu¹

¹ Department of Computer Science and Software Engineering

² Department of Information Systems,
University of Melbourne, Parkville, 3010, VIC, Australia

Abstract. One advantage of using the agent paradigm for software engineering is that the concepts used for high-level modelling, such as roles, goals, organisations, and interactions, are accessible to many different stakeholders. Existing research demonstrates that including the stakeholders in the modelling of systems for as long as possible improves the quality of the development and final system because inconsistencies and incorrect behaviour are more likely to be detected early in the development process. In this paper, we propose three changes to the typical requirements engineering process found in AOSE methodologies, with the aim of including stakeholders over the requirements engineering process, effectively using stakeholders as modellers. These changes are: withholding design commitment, delaying the definition of the system boundary, and delaying the stakeholder “sign-off” of the requirements specification. We discuss our application of these changes to a project with an industry partner, and present anecdotal evidence to suggest that these changes can be effective in maintaining stakeholder involvement.

1 Introduction

In software engineering, product and project stakeholders are a valuable resource for eliciting and validating requirements. Stakeholders are especially important for socio-technical systems, in which the interaction between people and technical systems can form behaviour outside of the control of the technology itself.

The agent paradigm recognises that most stakeholders are non-technical, so by using concepts such as roles and goals, which are palatable for most people, stakeholders can provide feedback on models early in the development process. As a result, artifacts in agent-oriented software engineering play a somewhat different role to other types of artifacts. As well as documenting the requirements engineers’ understanding of the domain, which requirements specifications typically do, they can also be used to encourage rich discussion between stakeholders, including requirements engineers.

Many requirements engineering processes, including those in agent-oriented software engineering methodologies, aim to define the interface and product features, and to precisely specify and validate these as early as possible in the development lifecycle. Our view is that, while making these decisions early has

benefits, premature commitment to certain solutions and definitions may discourage stakeholders that do not agree with or understand these decisions from participating in conversations with system developers. We advocate involving stakeholders in the development process for as long as possible, to continue engaging them in rich conversations that can help understand and define the system.

For engineering socio-technical systems, we propose small changes in the typical requirements engineering process found in software engineering (including AOSE) methodologies, with the aim of promoting conversation between stakeholders. The changes are based on results from existing research, which is discussed in Section 3. The proposed changes are:

1. Withholding design commitment by allowing inconsistencies and ambiguities early in the requirements engineering process. This allows different viewpoints of stakeholders to be represented, encouraging them into conversations for longer than they otherwise may. We are not the first authors to take this stance. For example, Easterbrook and Nuseibeh [7] discuss a framework with the purpose of allowing and dealing with different stakeholders' viewpoints. Paay et al. [19] suggest that withholding design commitment encouraged conversations between different stakeholders.
2. Delaying the definition of the system boundary. By defining the system boundary early in the process, some solutions may be eliminated before they can be discussed by the stakeholders, even though they may be more suitable than the remaining solutions.
3. Delaying the “sign-off” of requirements (or the end of the requirements engineering process) until the high-level agent design. That is, the requirements are considered only complete once we identify which agents are to be built and what their behaviour is to be. This is related to the second point, as it also helps to define the system boundary.

It is our view that these changes can be used in any agent-oriented development methodology, and are useful for breaking down barriers between stakeholders and software engineers, especially for social-technical systems. In Section 4, we present the application of these changes to an industry case study, and discuss the advantages and disadvantages that resulted from these changes. The goals of the paper are to present these processes to researchers and practitioners in agent-oriented software engineering in order to promote discussion and receive feedback on these ideas.

2 Agent-Oriented Requirements Engineering

With the agent paradigm increasingly becoming a popular and successful way for modelling complex systems [18], methodologies for agent-oriented software engineering have become an important research field. Several such methodologies have been proposed, such as Tropos [3], Prometheus [20], Gaia [30], INGENIAS [21], and ROADMAP [13].

The typical requirements engineering process in these methodologies involves the following steps³:

1. elicit requirements from the stakeholders on the project;
2. derive scenarios that specify typical usage of the system;
3. define the system boundary;
4. define the environment;
5. derive a goal model outlining the major goals of the system;
6. define the role descriptors for the roles that will help to achieve the system goals;
7. define the interaction model, which specifies how roles in the system will interact; and
8. iterate over steps 1-7 with stakeholders until a shared understanding of the system is reached.

Although agent methodologies do not discuss requirements sign off, they define the *software requirements specification* (SRS) as the combination of the system boundary, goal models, role, and interaction models. From this, we infer that the major stakeholders would sign off on these documents after step 8. This would form the basis of a contract for the system development to proceed.

Variations of these steps are possible; for example, the Gaia methodology defines preliminary version of the role and interaction models as requirements, and more detailed definitions as architectural design; and Prometheus defines interaction models as architectural design.

From this point in the development process, agent-oriented methodologies typically treat subsequent tasks as design-level, so stakeholder input would not be required. The tasks include defining the agent types in the system, which agent types will play which roles, the activities that the agents will perform (these activities will both fulfill the agent’s role and the goals related to that role), and implementing and testing the agents.

2.1 Modelling with Roles and Goals

The work in this paper builds mainly on the work of Sterling and Taveter [23]. Their work has focused on how to make high-level agent-oriented models palatable to non-technical stakeholders. This is achieved using role and goal models with a straightforward and minimal syntax and semantics.

Goal models are useful at early stages of requirements analysis to arrive at a shared understanding [14, 12]; and the agent metaphor is useful as it is able to represent human behaviour. Agents can take on roles associated with goals. These goals include quality attributes that are represented in a high-level pictorial view used to inform and gather input from stakeholders. For example, a role may contribute to achieving the goal “Release pressure”, with the quality goal “Safely”. We include such quality goals as part of the design discussion

³ Some methodologies do not strictly follow this process, but this is a good approximation of all methodologies.

and maintain them as high-level concepts while eliciting the requirements for a system. For this purpose the AOSE goal models have to be simple yet meaningful enough to represent the goals of social interactions.

Figure 1 shows the syntax employed by Sterling and Taveter, which we have used in our work. Goals are represented as parallelograms, quality goals are clouds, and roles are stick figures. These constructs can be connected using arcs, which indicate relationships between them. Figure 1 shows a high-level role and goal model from our industry project of an aircraft turnaround simulator. This system simulates the process of multiple aircraft landing at a single airport, allowing one to experiment with resource allocation. The goal *Aircraft Turnaround* is the highest-level goal, and the sub-goals below this contribute to fulfill the higher-level goal. The quality goal *Efficient* specifies that goal *Aircraft Turnaround* must be satisfied with the quality attribute *Efficient*. The roles play some part in bringing about the goal *Aircraft Turnaround*.

Fig. 1. An excerpt for the high-level goal on the aircraft turnaround project.

It is important here to note that the semantics described above is a complete definition of Sterling and Taveter’s goal models, leaving space for interpretation of the model. This helps to engage stakeholders who have no experience in agent modelling, and encourages round-table discussion between stakeholders and requirements engineers.

3 Changing the Agent-oriented Requirement Engineering Process

The changes presented in this paper are based on existing research in software engineering and interaction design, however, it is our view that the agent paradigm offers certain unique capabilities to the requirements engineering process that other paradigms do not. In this section, we motivate and justify our reasons for modifying the requirements engineering process, link this to existing literature that provides evidence to confirm our hypothesis, and discuss why the agent paradigm is particularly suited to these changes.

At first sight, delaying clear definitions seems antithetical or uncommon to the routines of software engineering, which is typically a structured process aimed at removing ambiguity and deriving clear definitions as early as possible in the development process. However, a body of literature that looks at software engineering from a social science perspective recognises that models and other documentation in software engineering have been used as a way to think through problems, to reach agreements, and to elaborate the needs of stakeholders in a different way than simply feeding into a formal process of modelling for system design [16, 4, 22]. For example, a goal and role model serves a different purpose for a designer than for a domain expert.

3.1 Withholding Design Commitment

The first change to the requirements engineering process is to withhold the commitment of system designs. By this, we mean holding off any particularly functional details of the system that fulfill the user requirements. At the early stages of requirements elicitation, we may not be able to clarify social concepts sufficiently to resolve uncertainty. For example, in a business domain, roles such as manager, researcher, and team leader can be well defined. However, in a social domain, roles may not be so straightforward to define. Consider trying to define the role of a grandparent, and the goals that role may want to achieve. As a result, we advocate that the social goals related to these concepts should be modelled ambiguously, even to the point where formal documents are written.

Quality requirements at the early stages of elicitation tend to be imprecise, subjective, idealistic and context-specific, as discussed by Jureta and Faulkner [14]. Garcia and Medinilla [9] describe high-level quality goals as a specific form of uncertainty that can be used as a descriptive complexity reduction mechanism and to model and discuss uncertainties in the environment. In our requirements elicitation process, we seek complexity reduction without losing the richness of the concepts themselves. Instead of eliminating uncertainty early in the process, we embrace it and withhold design commitment, at least until there is clarity and understanding between stakeholders of what it may mean to disambiguate [10].

High-level goals associated with activities can act as a point of reference for discussing the usefulness of design alternatives to achieve these goals instead of a decomposition into single requirements. The multi-agent paradigm offers benefits over other paradigms because the concepts used in modelling, such as roles, goals, and interactions, are part of every day language. Real organisations consist of roles, and specific people fill these roles each day, including stakeholders in a software engineering project. As such, stakeholders are familiar with these concepts, and are comfortable talking about them.

3.2 Delaying the Definition of the System Boundary

In many software engineering processes, the system boundary is defined before requirements analysis takes place. Often, this is one of the first agreements made between clients and software engineers.

Gause and Weinberg [11] found that natural subconscious disambiguation is one of the most common sources of requirements failure. In this situation, unrecognized or unconsciously assumed, incorrect meaning finds its way into the specification [2]. The problem is compounded by the fact that not only do software engineers consciously try to resolve uncertainty early in the process, before its impact on design is completely understood, they may also do this subconsciously. More importantly, checking the absence of requirements once we have a formal specification document is likely to be more difficult, because these documents are typically highly technical, and therefore less accessible to the stakeholders [15].

Once the boundaries of a system are defined, the focus of attention is within these boundaries; solutions beyond this boundary are no longer considered. Such a restriction discounts solutions that may be more suitable, and is more likely to result in some stakeholders losing interest in the project if their desired solution falls outside of these boundaries.

This does not imply that one should not be thinking about the system boundary. Specifically, all stakeholders should be aware of any other systems that may be used as part of the solution to the domain problem.

The multi-agent paradigm is well suited for such models, because high-level role and goal models can be discussed and modified without defining the system boundary, while still allowing all stakeholders to come to a shared agreement of what the entire socio-technical system will comprise.

3.3 Delaying the “Sign-off” of Requirements

The sign-off of the SRS often forms part of a contractual agreement between clients and developers. The SRS defines the external interfaces to a software system and provides a complete description of the extended behaviour of the software.

In the process of software engineering, the sign-off of a requirements specification is generally performed before any high-level design takes place. If left until after design commences, developers may unnecessarily waste time on design tasks, only to find the requirements have changed.

In the multi-agent domain, we advocate delaying the sign-off of the SRS by stakeholders until as late as possible before it impacts architectural design. This allows discussions to continue between stakeholders for a longer period. Furthermore, it also helps stakeholders to understand the proposed behaviour of the system, because role and goal models define motivation, not behaviour.

3.4 Discussion

The first two changes proposed in this section are not new in the social domain. Our work is consistent with results from researchers cited in the previous sections. As far as the authors are aware, the third change, delaying the sign-off, has not been investigated before.

While we present these three proposed changes as being separate changes, they are in fact, closely related. By not defining the system boundary, we are in fact withholding design commitment. Similarly, by not signing off on the SRS early, we are leaving open design decisions, thereby withholding design commitment.

These changes are presented separately because we view them as different tasks. Withholding design commitment is a general approach in which we do not take design decisions too early, but in general, the requirements elicitation process will run in the same order. However, the definition of the system boundary is a specific task that we aim to put later in the requirements engineering process. Typically, defining the system boundary is one of the first tasks performed in requirements engineering, and this is suitable for most business applications. However, for socio-technical systems, we see that a benefit in delaying the definition of the system boundary until after we fully understand the behaviour of the *entire* socio-technical system, including humans and external systems, not just the software system being built.

4 Experience

In this section, we present our experience on a project involving an industry partner. We discuss how the changes were achieved in an industry project, what effect they had on the project, and how other stakeholders responded to them.

4.1 The Project

The project is a joint project between the University of Melbourne and Jeppesen, a company that specialises in aeronautical services. The goal of the project is to construct simulation software for air traffic management using the agent paradigm as the modelling tool. The particular project on which we applied the modified requirements engineering process was a simulation of aircraft turnaround. This system simulates the process of multiple aircraft landing at a single airport, and how resources (including staff) could be allocated to efficiently turn around the aircraft, including re-stocking supplies, as well as cleaning, repairing, and maintaining the aircraft.

The major stakeholders of the project were our research team and a group of software engineers at Jeppesen who had no significant exposure to agent-oriented modelling in the past.

Figure 1 (in Section 2) shows part of the high-level role-goal model for the aircraft turnaround project. In this figure, the high-level goal of turning around

the aircraft is achieved by the four subgoals of preparing for arrival, servicing the aircraft, maintaining the aircraft, and preparing for departure. The roles of *Airline Staff* and *Airport Staff* in this figure are in fact *aggregate roles*; that is, they are sets of roles, such as aircraft maintenance engineers, cleaners, and airline crew, which are described in lower-level role-goal models. The *Manager* role is responsible for overseeing the entire turnaround and re-allocating resources if there is a delay in turning around one aircraft.

4.2 Withholding Design Commitment

The requirements Eli citation proceeded by our group being given an overview of the aircraft turnaround process, including the staff involved, and constructing a high-level goal and role model that represented our understanding of the system. These diagrams were improved and refined over a series of six round-table meetings with the stakeholders, in which the role and goal models were distributed to each stakeholder before a meeting, and were then used as shared artifacts to guide conversations. Over the course of these meetings, other models including the interaction models, environment models, and agent types were progressively introduced as we gained further understanding of the system.

Withholding design commitment was achieved by basing conversations between stakeholders on the role-goal models and using the role-goal models as a facilitator to open up the discussion. In this regard, the goal models took a similar role as the guiding rules described by Tjong et al. [24], whose aim is to detect uncertainties in order to trigger questions to be asked of the client.

The role and goal models were helpful in triggering communication about the specific challenges of the domain, and for identifying missing parts of the system. For example, one stakeholder commented from a single glance at the high-level goal model that air traffic controllers play a role in aircraft turnaround, and this induced discussion about how the system should handle new traffic entering the airport. In subsequent iterations, the air traffic controller role was deemed unnecessary for the system and was dropped, but changes related to this remained.

Our experience indicates that having models evolve over time lead to a clearer solution, as early concerns regarding concepts such as resources were delayed without jumping to a pre-conceived solutions. Later in the development process, successive versions of the models were used as a reminder to the design decisions that were made. This gave the research team something to fall back on when discussions started to get too complex for some stakeholders or drifting off from original high-level goals. The example of the air traffic controller role illustrates this, in which the models were updated to reflect this role, but even after its removal, parts of the model related to it remained. This is consistent with the findings described by MacLean and Bellotti [16].

Our industry partners are comfortable with the role and goal models, although this is perhaps to be expected as they are software engineers. However, Paay et al. [19] have used role and goal models as shared artifacts in the social-

technical domain with non-technical stakeholders such as ethnographers to similar effect.

4.3 Including Agent Types as an SRS

We delay the system boundary definition and the SRS sign-off using the same technique: by leaving both until the high-level design.

The major divergence we take from the typical AOSE methodology is to include the agent types, including the activities they perform, as part of the SRS. As discussed in Section 2, methodologies typically use roles, goal, and interaction models as requirements, while agent types are part of the architectural design.

In this project, the SRS consisted of the role and goal models, the interaction models, the environment model, and the agent types. Combining the environment model and the agent types defines the functionality of the system, while the role and goal models help to motivate this functionality. For this particular simulation system, there was a one-to-one mapping between roles and agents.

Figure 2 shows part of the agent type specification for the *Engineer* agent, which is responsible for performing routine and non-routine maintenance on the aircraft. The agent type specification includes which activities the agent will undertake in order to fulfil its responsibilities.

Signing-off on the SRS We believe that roles, goals, and interactions do not provide sufficient detail to define system behaviour. While role and goal models specify the goals that the system will achieve, and the roles (and their responsibilities) that will help to achieve them, they do not define functionality; that is, how the system will behave to achieve these goals. For example, the model in Figure 1 specifies the goals that need to be achieved to turnaround the aircraft. Role descriptors for the three roles in this figure outline the responsibilities to ensure the turnaround goals are achieved. However, this does not define which activities will be performed to achieve the goals. In some cases, one can extrapolate the activities from the responsibilities and goals, but this is not always the case.

Our approach of including the environment model and agent types, including activity descriptions and their effect on the environment, specifies the behaviour of the system. From Figure 2, one can see that activity descriptors are similar to the functional requirements that one would find in a non-agent-based SRS, and it is at this point that the major stakeholders will be able to sign-off on the models.

Figure 3 presents a possible template for an agent-based SRS, based on Wiegers' SRS template [26]. Using a template leads to requirements being presented in a consistent manner across different projects, however, we acknowledge the need to be flexible with specifications depending on the system.

Our template differs from Wiegers' template mainly by emphasising the importance of motivations (using role and goal models) and the environment, which are central to the agent paradigm. Wiegers considers both the purpose of the

Name:	Engineer														
Description:	Play the role of <i>Engineer</i> by performing routine and non-routine aircraft maintenance.														
Activities:	<table> <tr> <td>Activity name:</td><td>Routine maintenance</td></tr> <tr> <td>Trigger:</td><td>Informed by ground staff of the aircraft ID of the aircraft that is ready for maintenance</td></tr> <tr> <td>Precondition:</td><td>Wheel chocks of the aircraft ID are in position</td></tr> <tr> <td>Tasks:</td><td> 1. Perform the routine maintenance on the specified aircraft 2. Inform <i>Pilot</i> of the aircraft ID, and that routine maintenance is complete on the aircraft </td></tr> <tr> <td>Postcondition:</td><td>Aircraft with the specified ID is safe to fly</td></tr> <tr> <td>Activity name:</td><td>Non-routine maintenance</td></tr> <tr> <td></td><td>...</td></tr> </table>	Activity name:	Routine maintenance	Trigger:	Informed by ground staff of the aircraft ID of the aircraft that is ready for maintenance	Precondition:	Wheel chocks of the aircraft ID are in position	Tasks:	1. Perform the routine maintenance on the specified aircraft 2. Inform <i>Pilot</i> of the aircraft ID, and that routine maintenance is complete on the aircraft	Postcondition:	Aircraft with the specified ID is safe to fly	Activity name:	Non-routine maintenance		...
Activity name:	Routine maintenance														
Trigger:	Informed by ground staff of the aircraft ID of the aircraft that is ready for maintenance														
Precondition:	Wheel chocks of the aircraft ID are in position														
Tasks:	1. Perform the routine maintenance on the specified aircraft 2. Inform <i>Pilot</i> of the aircraft ID, and that routine maintenance is complete on the aircraft														
Postcondition:	Aircraft with the specified ID is safe to fly														
Activity name:	Non-routine maintenance														
	...														
Environment considerations:	1. Aircraft 2. Aircraft information 3. Flight schedule 4. Aircraft gate number 5. Staff schedule														

Fig. 2. Agent type specification for the *Engineer* agent.

system and the environment, but these are secondary in the SRS. In addition, the functional requirements section from Wiegers' template is replaced by agent types, which define the behaviour.

A sign off is an agreement that overall goals are important, and that the defined system will achieve these goals. In our project, all stakeholders came to a solution that all were satisfied with. We see this as a benefit in itself.

Furthermore, the stakeholders commented that the behaviour of the system was clearer when the agent types were included, even though the mapping from roles to agents was one-to-one. This is perhaps partly due to the similarity between activities and functional requirements, but the stakeholders commented that this was due to the fact that they were able to make a clear judgement as to whether the behaviour fulfilled their expectations. In our view, this justifies the decision to include the agent types in the SRS.

Defining the System Boundary Including agent types in an SRS has a second effect: it completely defines the system boundary. Role and goal models define the entire socio-technical system, with no commitment to which roles will be played by which agents. As Cheng and Atlee [5] discuss, integrated systems

Title information

Revision history

Table of contents

1 Introduction

- 1.1 Purpose
- 1.2 Intended audience
- 1.3 Project scope
- 1.4 Definitions, acronyms, and abbreviations
- 1.5 References

2 Product Description

- 2.1 High-level level motivation model
- 2.2 User classes
- 2.3 Product features
- 2.4 Design constraints
- 2.5 Assumptions

3 Goal models and motivational scenarios

- 3.1 Motivational scenarios
- 3.2 Goal models

4 Role and organisational models

- 4.1 Organisational model(s)
- 4.2 Role 1
- 4.3 Role 2
- etc ...

5 Domain/Environment model

- 5.1 Physical environment
- 5.2 Virtual Environment
- 5.3 Environment perspective
- 5.4 Overall interaction

6 Agents types and interaction models

- 6.1 Interaction models
- 6.2 Agent type 1
- 6.3 Agent type 2
- etc ...

7 Scenarios

- 7.1 Scenario 1
- 7.2 Scenario 2
- etc ...

8 External interfaces

- 8.1 User interfaces
- 8.2 Hardware interfaces
- 8.3 Software interfaces

9 Endorsement

- 9.1 Sign-off

Fig. 3. A software requirements specification template using Sterling and Taveter's models [23].

pose problems in defining the system boundary, which can be solved by assigning responsibilities to different parts of the system, including the software system being constructed, human operators/users, and external systems. Our notion of a system boundary is exactly this: by describing the responsibilities of roles in the entire system, we can define the system boundary by specifying which agents will fulfil which roles, whether these agents are software agents, humans, or external systems.

For example, consider the organisation model in Figure 4, which describes the relationships between the roles in the system. A possible mapping between roles and agents is one in which software agents play all of the relevant roles, making the system a complete simulator of the turnaround process. Alternatively, we can define another system boundary in which the *Manager* role is played by a human, and thus the dotted arrows in Figure 4 define the interactions between the user and the software. One can see that assigning one role to a human instead

of an agent changes the system and its interface greatly. In the first instance, the system is a complete simulation of the aircraft turnaround process. In the second instance, the result is an interactive system in which managers are able to assess different resource allocation mechanisms.

Fig. 4. The organisational model for the ATS system.

In this project, the system boundary was left undefined for most of the requirements elicitation process. The stakeholders were comfortable with the lack of a system boundary, and this was not explicitly mentioned to them during the requirements elicitation. However, as software engineers themselves, they did not see any great benefit for this project, because they felt only one system boundary was sensible. However, they also did not find that it was detrimental to the project. We did not find that delaying the definition of the system boundary had any adverse effects on the progression of the system, although this was not a controlled experiment. In addition, we found that conversations about the system, including details about roles and goals, continued after the agent types had been assigned, due to the system functionality becoming clearer.

To our group, the benefits of not defining a system boundary are illustrated by the project. The system was intended to be a simulation of the air traffic turnaround domain, with all roles, including those in Figure 1, being played by software agents (the first boundary in the previous paragraph). One discussion

that took place late in the requirements elicitation process indicated that there may have been scope for the system boundary to be changed to the second boundary, in which the *Manager* role is partly played by a human. Had the system boundary been defined at the start of the requirements elicitation, this discussion may not have taken place.

5 Related Work

Guizzardi and Pereni [12] have also recognised the importance of stakeholder involvement. Like us, they consider the goals of all stakeholders, and the interdependencies between these goals, as an initial step in understanding requirements. Yu [27–29] advocates the agent-oriented paradigm as a tool for helping to establish the *why* of a system, which helps stakeholders to understand the problem at hand. Similar to us, Yu uses high-level motivation models, in this case, specified in i^* , to share understanding between stakeholders. The i^* models contain significantly more information than our motivation models, including concepts such as activities, resources, and dependencies between all of these. We explicitly aim to reduce the number of concepts and the amount of syntax to keep models simple. Yu offers no specific *techniques* for engaging stakeholders, as the focus of the work appears to be on the tools and notations for recording motivations.

We are not the first authors to identify that high-level conceptual models in agent methodologies are not sufficient to define behaviour. Ferber et al. [8] identify two approaches for specifying behaviour of a multi-agent system. The first approach resembles that of specifying individual requirements of a system, with the addition of nominating the agent that is undertaking each task, thus specifying an observer’s view of system behaviour. The second approach involves assigning behaviour to *role instances*, and specifying behaviour from the viewpoint of the individual instance. Interactions between roles are specified as behaviour similar to protocol specifications. This approach is closer to our approach than the first approach, however, we feel that the intermediate representation between roles and agents is unnecessary.

The KAOS methodology [6] defines the behaviour of systems using *agent/action* definitions. These are similar to our agent types, in that they define the agent and the actions that the agent can perform. When applying the KAOS methodology to an example of a meeting schedule, van Lamsweerde et al. [25] comment that the last stages of the *goal elaboration* process, in which goals are refined and analysed, “were performed in parallel with the agent/action identification and goal operationalisation”. This provides further evidence that committing to some agent or activity design is necessary to define behaviour.

The Prometheus methodology [20], like KAOS, does not consider roles as part of the elicitation, modelling, or specification process. Similar to us, they identify that functionality must be considered to define behaviour. A Prometheus specification contains the system goals, but with no indication of the roles that achieve

them. *Functionalities*, which are natural language descriptions of behaviour, are used to define the system behaviour.

The agent paradigm has been applied in many industry systems. Munroe et al. [18] and Belecheanu et al. [1] describe several applications of agent technology in industry, and discuss key issues that agent proponents face to have their approaches accepted widely into industry.

Maiden et al. [17] is the most closely related work to ours of which the authors are aware. They use several different methodologies and notations, including the agent-oriented i^* methodology, to capture the requirements of an air-traffic control system. The purpose of using different notations and methodologies is to capture the differing viewpoints of the stakeholders within the project. Maiden et al. do not offer any particular methods for engaging stakeholders, however, using different approaches has a positive side effect of being able to cross-validate the different models against each other.

6 Conclusions and Related Work

AOSE models are useful as a shared artifact for communication between stakeholders and software engineers. We find that using the agent-oriented models of Sterling and Taveter [23] as part of requirements elicitation allows meaningful conversations between all stakeholders about abstract concepts, with goals as the catalyst. The role of the goal models is not simply to lead to the development of a system, but also as a way to think through problems and to reach agreements. By making these accessible to all stakeholders, and by keeping stakeholders involved in discussions as long as possible in the requirements elicitation process, we aim to increase the quality of requirements specifications.

In this paper, we proposed three changes to the typically AOSE requirements engineering process that we believe help to engage stakeholders: 1) withholding design commitment; 2) delaying the definition of the system boundary; and 3) delaying the sign-off of the SRS to be as late as possible without affecting system development.

Our experience with an industry partner suggests that not committing to a specific design solution early in the requirements elicitation gave the team an opportunity to further explore and understand the specific challenges related to the high-level goals of the socio-technical system.

We propose delaying the definition of the system boundary and the signing-off of the requirements by including the agent types as part of the SRS. As far as we know, we are the first authors to consider this, rather than including these as part of the architectural or detailed design. By defining which agents will play which roles, we define the behaviour of the system, and implicitly define the system boundary. The experience with our industry partner indicates that this decision is justified.

References

1. R.A. Belecheanu, S. Munroe, M. Luck, T. Payne, T. Miller, P. McBurney, and M. Pěchouček. Commercial applications of agents: Lessons, experiences and challenges. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1555–1561. ACM Press, 2006.
2. D. Berry, E. Kamsties, and M. Krieger. From contract drafting to software specification: Linguistic sources of ambiguity - a handbook version 1.0, 2000.
3. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
4. G. Button and W. Sharrock. Occasioned practices in the work of software engineers. In M. Jirotko and J. Goguen, editors, *Requirements Engineering: Social and Technical Issues*, pages 217–240. Academic Press, 1994.
5. B. Cheng and J. M. Atlee. Research directions in requirements engineering. In L. Briand and A. Wolf, editors, *Proceedings of the International Conference on Software Engineering*, pages 285–303, 2007.
6. A. Dardenne, A. Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1-2):3–50, 1993.
7. S. Easterbrook and B. Nuseibeh. Using viewpoints for inconsistency management. *Software Engineering Journal*, 11(1):31–43, 1995.
8. J. Ferber, O. Gutknecht, C.M. Jonker, J.P. M
uller, and J. Treur. Organization models and behavioural requirements specification for multi-agent systems. In Y. Demazeau and F. Garijo, editors, *Multi-Agent System Organisations. Proceedings of the 10th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 1–19, 2001.
9. A. Garcia and N. Medinilla. The ambiguity criterion in software design. In *International Workshop on Living with Uncertainties*. ACM, 2007.
10. D. Gause. User driven design – the luxury that has become a necessity, a workshop in full life-cycle requirements management. In *ICRE 2000, Tutorial T7*, 2000.
11. D. Gause and G. Weinberg. *Exploring Requirements: Quality Before Design*. Dorset House Publishing Co., Inc., New York, NY, USA, 1989.
12. R. Guizzardi and A. Perini. Analyzing requirements of knowledge management systems with the support of agent organizations. *Journal of the Brazilian Computer Society (JBACS)-Special Issue on Agents Organizations*, 11(1):51–62, 2005.
13. T. Juan, A. Pearce, and L. Sterling. ROADMAP: Extending the Gaia methodology for complex open systems. In *Proceedings of the First Int. Conf. on Autonomous Agents and Multi-Agent Systems*, pages 3–10. ACM Press, 2002.
14. I. Jureta and S. Faulkner. Clarifying goal models. In J. Grundy, S. Hartmann, A. Laender, L. Maciaszek, and J. Roddick, editors, *ER (Tutorials, Posters, Panels & Industrial Contributions)*, volume 83 of *CRPIT*, pages 139–144, 2007.
15. E. Kamsties, D. Berry, and B. Paech. Detecting ambiguities in requirements documents using inspections. In *Proceedings of the First Workshop on Inspection in Software Engineering (WISE’01)*, pages 68–80, 2001.
16. A. MacLean, V. Bellotti, and R. M. Young. What rationale is there in design? In D. Diaper, D. J. Gilmore, G. Cockton, and B. Shackel, editors, *Proceedings of the 3rd Int. Conf. on Human-Computer Interaction*, pages 207–212, 1990.
17. N. Maiden, S. Jones, S. Manning, J. Greenwood, and L. Renou. Model-driven requirements engineering: Synchronising models in an air traffic management case study. In *Proceedings of the Conference on Advanced Information Systems Engineering*, volume 3084 of *LNCS*, pages 368–383. Springer-Verlag, 2004.

18. S. Munroe, T. Miller, R. Belecheanu, M. Pěchouček, P. McBurney, and M. Luck. Crossing the agent technology chasm: Lessons, experiences and challenges in commercial applications of agents. *Knowledge Engineering Review*, 21(4):345–392, December 2006.
19. J. Paay, L. Sterling, F. Vetere, S. Howard, and A. Boettcher. Engineering the social: The role of shared artifacts. *International Journal of Human-Computer Studies*, 67(5):437–454, 2009.
20. L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A practical guide*. John Wiley and Sons, August 2004.
21. J. Pavón and J. Gómez-Sanz. Agent oriented software engineering with INGENIAS. In *Multi-Agent Systems and Applications III*, volume 2691 of *LNCS*, pages 394–403. Springer, 2003.
22. D. Randall, J. Hughes, and D. Shapir. Steps toward a partnership: ethnography and system design. In M. Jirotko and J. Goguen, editors, *Requirements Engineering: Social and Technical Issues*, pages 241–254. Academic Press, 1994.
23. L. Sterling and K. Taveter. *The Art of Agent-Oriented Modelling*. MIT Press, 2009.
24. S. F. Tjong, M. Hartley, and D. Berry. Extended disambiguation rules for requirements specifications. In C. Alves, V. Werneck, and L. Marcio Cysneiros, editors, *Proceedings of Workshop in Requirements Engineering*, pages 97–106, 2007.
25. A. Van Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 194–203. IEEE Computer Society, 1995.
26. K.E. Wiegers. *Software requirements*. Microsoft Press, second edition, 2003.
27. E. Yu. Modeling organizations for information systems requirements engineering. In *Proceedings First IEEE International Symposium on Requirements Engineering*, pages 34–41. IEEE, 1993.
28. E. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, page 226. IEEE Computer Society, 1997.
29. E. Yu. Agent-oriented modelling: software versus the world. In *Agent-Oriented Software Engineering II*, volume 2222 of *LNCS*, pages 206–225. Springer, 2002.
30. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering Methodology*, 12(3):317–370, 2003.