2014

Requirements elicitation and specification using the agent paradigm: the case study of an aircraft turnaround simulator

Tim Miller
University of Melbourne, tmiller@unimelb.edu.au

Bin Lu
University of Melbourne, lbin@unimelb.edu.au

Leon Sterling
University of Technology, Melbourne, lsterling@swin.edu.au

Ghassan Beydoun
University of Wollongong, beydoun@uow.edu.au

Kuldar Taveter
Tallinn University of Technology, kuldar.taveter@ttu.ee

Publication Details

Miller, T., Lu, B., Sterling, L., Beydoun, G. & Taveter, K. (2014). Requirements elicitation and specification using the agent paradigm: the case study of an aircraft turnaround simulator. IEEE Transactions on Software Engineering, 40 (10), 1007-1024.

Requirements elicitation and specification using the agent paradigm: the case study of an aircraft turnaround simulator

Abstract
In this paper, we describe research results arising from a technology transfer exercise on agent-oriented requirements engineering with an industry partner. We introduce two improvements to the state-of-the-art in agent-oriented requirements engineering, designed to mitigate two problems experienced by ourselves and our industry partner: (1) the lack of systematic methods for agent-oriented requirements elicitation and modelling; and (2) the lack of prescribed deliverables in agent-oriented requirements engineering. We discuss the application of our new approach to an aircraft turnaround simulator built in conjunction with our industry partner, and show how agent-oriented models can be derived and used to construct a complete requirements package. We evaluate this by having three independent people design and implement prototypes of the aircraft turnaround simulator, and comparing the three prototypes. Our evaluation indicates that our approach is effective at delivering correct, complete, and consistent requirements that satisfy the stakeholders, and can be used in a repeatable manner to produce designs and implementations. We discuss lessons learnt from applying this approach.

Disciplines
Engineering | Science and Technology Studies

Publication Details
Miller, T., Lu, B., Sterling, L., Beydoun, G. & Taveter, K. (2014). Requirements elicitation and specification using the agent paradigm: the case study of an aircraft turnaround simulator. IEEE Transactions on Software Engineering, 40 (10), 1007-1024.

Requirements engineering using the agent paradigm: a case study of an aircraft turnaround simulator

Tim Miller, *University of Melbourne*
Bin Lu, *University of Melbourne*
Leon Sterling, *Swinburne University of Technology*
Ghassan Beydoun, *University of Wollongong*
Kuldar Taveter, *Tallinn University of Technology*

Abstract—In this paper, we describe improvements to our previous work on agent-oriented requirements engineering. The aims of these improvements are to mitigate two problems experienced by ourselves and our industry partner: (1) the lack of systematic methods for agent-oriented requirements elicitation and modelling; and (2) the lack of prescribed deliverables in agent-oriented requirements engineering. We discuss the application of our new approach to an aircraft turnaround simulator built in conjunction with an industry partner, and show how agent-oriented models can be derived and used to construct a complete requirements package. Our experience indicates that our approach is effective at delivering correct, complete, and consistent requirements that satisfy the stakeholders. We discuss what we have learnt from applying this approach.

1 Introduction

Evidence suggests that incorrect and poorly-specified requirements are a major cause of software project failure, with two major contributing factors being requirements complexity and a lack of stakeholder input [9], [14]. Stakeholders are often not capable of articulating their requirements fully at the beginning of a project. Early-stage requirements tend to be imprecise, subjective, idealistic and context-specific [18]. In our earlier work [23], [28], we used an incremental approach to requirements modelling to engage stakeholders, acknowledging that requirements elicitation is better supported with a spiral process that narrows down the design and implementation choices as it progresses. Instead of eliminating uncertainty early, we embrace it and withhold design commitment, at least until there is clarity and understanding between stakeholders of what it may mean to disambiguate [11]. Committing early to requirements can forgo an opportunity to properly disambiguate them [13].

Over the past several years, we have worked with several industry and academic partners, including Adacel Technologies[1], Lockhard[2], and Jeppesen[3], to improve requirements engineering using agents as the central paradigm. Our industry partners face the problem of eliciting and recording requirements of complex systems that contain many interacting parts, and many interactions between different actors. One such product is a large-scale system for the air traffic domain that allows simulation of complex trade-offs between interacting actors in a socio-technical system. These partners have identified that the agent-oriented paradigm is a natural metaphor for modelling the social considerations in their systems, emphasising the "why" questions that can help in requirements elicitation, and producing models that are more accessible to their non-technical stakeholders [3], [25], [23], [35].

While existing agent-oriented requirements engineering approaches have matured over the past decade, our current industry partner identifies three major drawbacks with existing work, including our own:

1) Eliciting and recording relevant information in agent-oriented models is a non-trivial problem. Existing methodologies do not describe, in a systematic and prescriptive manner, what information to elicit and how to record. Thus, the modelling remains far more art than engineering. In our previous projects, we have observed experienced software engineers modelling the system in a way with which they are familiar, but using agent-related terms; for example, modelling the system using the object-oriented paradigm, but using "agent" in place of "object"; therefore negating any advantage of using the agent paradigm.
2) Many existing modelling notations contain considerable detail, such as cardinality constraints and relationship types, much of which is relevant to developers, but which does not adequately engage stakeholders. This can limit the ability for requirements engineers to decipher what is important, especially early in the requirements phase, when a vast amount of information is presented.
3) Existing methodologies do not prescribe how to produce requirements specifications using agent-oriented models. In our collaborative work, our industry partner could not see how to interpret agent-oriented models as requirements specifications that could be implemented and used for verification. Agent-oriented methodologies usually focus on the agent-oriented aspects, but overlook other aspects of software engineering, such as useful deliverables. Typically, a set of models is considered as a deliverable, providing little support for defining artifacts such as software requirements specifications, business vision documents, and system design descriptions, even

1. http://www.adacel.com/
2. http://www.lochard.com/
3. http://www.jeppesen.com/

though these are vital to industrial practice. This lack of support is unsurprising because publishing papers on deliverable formats may not be considered a valid scientific contribution.

As part of a larger grant funded by the Australian Research Council and Jeppesen, a company that develops software for the aviation and aerospace industries, we are exploring how agent-oriented models can be used in conjunction with a mature piece of software that needs to be maintained and enhanced. In a previous article [23], we identified several techniques for engaging stakeholders in the requirements engineering process. We use lightweight agent-oriented models to represent the roles, goals, and motivations of the greater socio-technical system, and to develop a shared understanding of these goals between the project stakeholders. Further, we advocate several strategies for delaying design decisions with the aim of encouraging stakeholder involvement.

This paper offers the following two contributions that build on both our and other researchers' existing work in agent-oriented requirements engineering.

1) **Requirements elicitation, analysis, and modelling:** In Section 4, we present a systematic and repeatable approach for eliciting, analysing, and modelling the requirements of a system in an agent-oriented manner. This approach prescribes a list of questions that must be answered by the project stakeholders, and further prescribes how to link the answers directly to agent-oriented models. In particular, this approach aims to place requirements engineers into the "agent mindset", to gain the full benefit of agent-oriented modelling.
   Our approach improves on existing agent-oriented requirements elicitation approaches by providing a more prescriptive and systematic way for software engineers to elicit information and construct models from these.
2) **Requirements specification:** In Section 5, we prescribe how to create a *software requirements specification* (SRS) built on agent-oriented models, which emphasises those aspects of the agent paradigm that are important for understanding the system. The end result is not (necessarily) a specification of a multi-agent system, but a specification of a system that uses the agent paradigm to describe motivation, structure, behaviour, and interaction.
   Our requirements specification template improves on previous work in agent-oriented software engineering by advocating the inclusion of agent types in requirements specifications.

To evaluate these contributions, we engineered a prototype of an aircraft turnaround simulator in conjunction with our industry partner. We used the ROADMAP/AOR agent-oriented development methodology [28], [17], but we believe that the contributions in this paper are general enough to be used with other agent-oriented methodologies.

The project and simulator are described in Section 2. An introduction to our previous work is discussed in Section 3. Applying our approach to the case study, we produced a requirements package that the client considered correct, complete, and consistent, and was developed into a prototype system. Section 6 discusses the verification and validation approach that we used for the system, which is based on some of our earlier work [19]. Section 7 discusses the most important lessons that we learnt from the evaluation.

2 Case study: Aircraft turnaround simulator (ATS) system

In this section, we describe the running case study, built in collaboration with our industry partner, used in this paper.

2.1 The research project

This case study is part of a larger joint project between the University of Melbourne and Jeppesen, a company that specialises in aeronautical services. One of Jeppesen's flagship products is the *Total Airspace and Airport Modeller* (TAAM), which allows modelling and simulation of airports and the surrounding airspace to help with decision making regarding infrastructure and operations. This product is a large-scale complex system that contains many interacting parts, and many interactions between different actors. The current event-based model and implementation is proving difficult to understand, maintain and enhance due to its size and complexity.

Jeppesen identified that using agent-oriented methods may help to manage the complexity and scale of their systems. The agent metaphor provides a natural and suitable way to represent a socio-technical system consisting of actors, their interactions, and their trade-offs. A major goal of the project is knowledge transfer between the University of Melbourne research team, and the software engineers at Jeppesen, as to how requirements should be elicited, modelled, and analysed.

As part of their own assessment of agent-oriented methods, Jeppesen identified that existing work does not provide a systematic and prescriptive method for eliciting requirements. Their engineers found it difficult to know where to start the process of developing agent-oriented models for requirements. In addition, during this project, it became clear that software engineers at Jeppesen could not see how agent-oriented models could be interpreted as requirements.

The core of the Jeppesen team on the project consists of three members with qualifications in physics, biophysics, and computer science respectively. All are familiar software engineering, but none have any significant prior experience in applying agent-oriented software engineering principles.

2.2 Aircraft turnaround simulation

As part of the knowledge transfer in the project, we have undertaken a smaller project, in which we aim to develop a simulator for aircraft turnaround using agent-oriented methods. This particular system was chosen because it is complex enough to demonstrate many aspects of the agent-oriented paradigm, but also manageable for our research group, who are not experts in aviation.

The system developed as part of the project is called the *Aircraft Turnaround Simulator* (ATS) system. The ATS 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, cleaning, repairing, and maintaining the aircraft.

The purpose of the system is to allow a user to evaluate different resource allocations mechanisms at airports. As such, the user should be able to set up parameters that specify the properties of the airport, the resources available, the schedules of staff, and the schedules of the arriving and departing aircraft. The system must produce reports describing the start and end points of all activities undertaken by staff, which can be used to assess the efficiency of allocation mechanisms.

3 Agent-oriented modelling

To model requirements, we use the notation of Sterling and Taveter [28]. Their work has focused on how to make high-level agent-oriented models palatable to non-technical stakeholders, and to carry these through to design and implementation. This is achieved using models with a straightforward and minimal syntax and semantics. In this section, we briefly describe the models that are relevant for this paper.

Goal models are useful at early stages of requirements analysis to arrive at a shared understanding [18], [15]; 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 and maintain them as high-level concepts while eliciting the requirements for a system.

Role models describe the capacities or positions that facilitate the achievement of goals. Roles have *responsibilities*, which outline what an agent playing the role must do to achieve the related goals, and *constraints*, which determine the conditions that must be considered when trying to achieve goals. Figure 1 defines the notation employed by Sterling and Taveter in their role and goal models. 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.

Goals are based on motives, and describe an intended state of the environment. Goals can consist of sub-goals.

Quality goals are non-functional (or quality) goals. These are sometimes referred to as *soft goals*.

Roles are the capacities or positions that are required for achieving of goals. Roles are played by agents, which can be humans or artificial.

Fig. 1: Sterling and Taveter's notation for goal modelling.

Organisation models represent the relationships between roles in a system. Zambonelli et al. [36] define several relationships between pairs of roles, and these definitions are widely accepted in the literature. In our work, there are three relationships that we have found useful: *control*, in which one role delegates responsibilities to another; *peer*, in which either role can delegate responsibilities to another; and *benevolence*, in which a role offers to fulfil responsibilities for another if it is in the offering role's interests.

Domain or *environment models* describe the relevant entities and relationships in an environment that the system operates. These can be represented in any suitable modelling language.

Agent and acquaintance models define the agents that will play the roles in the system, and the interaction pathways between the agents (similar to organisation models). Agents can be human or non-human, such as software or robotic.

Behavioural models and *knowledge models* specify the behaviours of the agents, and the knowledge that is required by the agents to perform their behaviours.

Interaction models represent communicative and physical interactions between the agents involved; that is, the activities in which two or more agents participate.

While the case study in this paper uses Sterling and Taveter's models, one can relate the approach to other agent-oriented notations and methodologies by identifying which models fit in the particular *viewpoints*. Sterling and Taveter [28] present the viewpoints of four other methodologies: Gaia [37], MaSE [7], Tropos [3], and Prometheus [25].

4 Elicitation, modelling, and analysis

In our experience working with industry and academic partners, we have found that a major barrier to using the agent paradigm to engineer requirements is the mindset of the requirements engineer. For example, those people familiar with object-oriented modelling will naturally design an "agent" system in which agents are directly mapped to objects, and messages are directly mapped to method calls, thus eliminating any advantage of using the agent paradigm.

In this section, we propose an approach for agent-oriented requirements elicitation, analysis, and modelling as a series of questions aimed to identify what needs to be elicited, and to analyse the elicited information, producing agent-oriented requirements models of the system. The questions naturally lead the people answering them to think of the system in terms of roles, goals, and interactions — helping the requirements engineers to get into the "agent mindset".

It is important to note that these questions are not necessarily to be used a interview questions, although interviews can form part of the input. The questions form a checklist, but one in which items are posed as questions, rather than items. These questions can be answered using techniques such as domain analysis, introspection, or group meetings. The questions and corresponding rules offer a prescriptive approach to producing models, and our experience is that many can be answered without having to present the question to a stakeholder.

The process followed is a straightforward elicitation process of identifying the problem and proposing a solution, involving the following steps:

Step 1: identify the problem, root causes, and stakeholders;
Step 2: develop a shared understanding of the *existing* system used to solve the problem, modelled using roles, goals, and interactions;
Step 3: identify a solution that uses the metaphor of a new staff position solving the problem; and
Step 4: specify the agent types that will play the roles in the system, generally with the new staff position being partially filled by the new software.

4.1 Engaging stakeholders in elicitation and modelling

In the ATS project, we elicited requirements using a combination of domain analysis, introspection, and round-table discussions with the stakeholders These round-table discussions allow the models, and as a result, our understanding, to evolve over time. They are also one key to engaging the stakeholders, and to not committing to a design too early. We use the term "round-table" instead of "group meeting" to differentiate the standard process of requirements engineers asking questions and taking notes, to our process of the many different stakeholders deriving models during the discussions.

While some modelling was performed outside of these meetings, this was to produce models that could be used as a starting point for subsequent discussions, which were then modified in the round-table discussions.

4.2 Our approach to systematic elicitation, analysis, and modelling

4.2.1 Identifying the problem, root causes, and stakeholders: a business vision

The first step is to identify the problem, the root causes of the problem, and the stakeholders. These properties of the project are recorded in what our industry partner terms a *business vision* document. The aim of this artifact is to reach a shared agreement of the problem, and also a high-level agreement of a solution space.

This step is standard in many projects, however, one difference to other approaches is that we use goal models to represent the motivations of the *project*, as well as the socio-technical system in which the software system will reside.

Figure 2 presents the project motivation model for the ATS system. The goal of the two stakeholders is to develop an aircraft turnaround simulator. Three quality goals were noted. First, the product must be developed using the agent-oriented paradigm. While this may seem as a unnecessary constraint on the system design, it was an important *project* quality goal because the purpose of the project was knowledge transfer in the area of agent-oriented software engineering. Also, the product must be testable and usable. These are two important quality goals for all projects undertaken by our industry partner. At this level, measurable definitions of testable and usable are not important.

In addition to the project motivation, we also derive a high-level model of the system motivation. This outlines the goals of the entire system, not just of the software to be built.

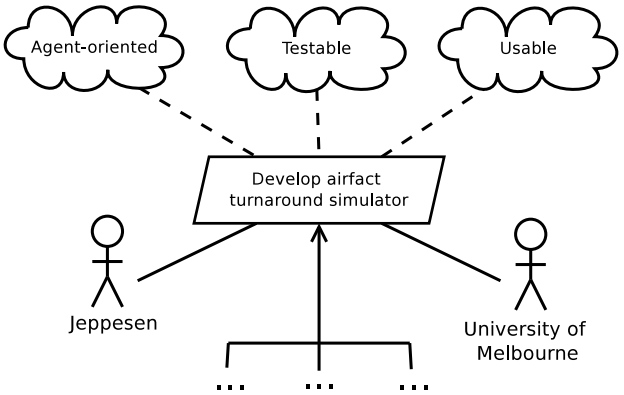

Fig. 2: An excerpt of the project motivation model for the ATS system.

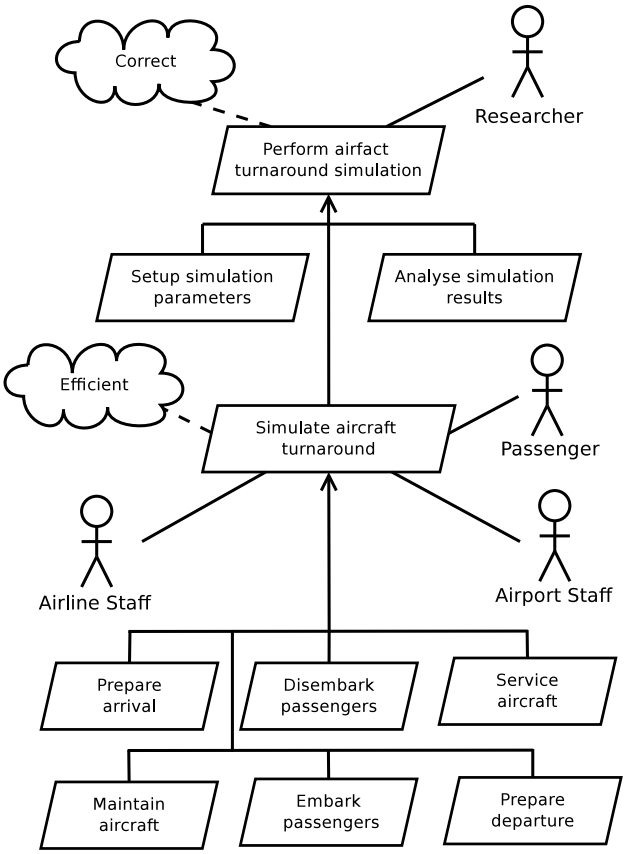

Fig. 3: The high-level motivation model for the ATS system.

These artifacts, including the motivation models, are signed off by the client (or major stakeholders). Our business vision documents have the structure outlined in Figure 4.

4.2.2 Understanding the current system

The second step is to understand the current system being employed to solve the problem; perhaps a manual system, or other software.

Zave and Jackson [38] argue the importance of understanding an entire system, including the environment in which a piece of software will operate. We agree that it is important to first understand the motivations of the existing socio-technical

Title information

Revision History

1 Introduction

2 Project Brief

2.1 *Problem*: description of the problem.
2.2 *Root causes*: root causes of the problem.
2.3 *Project stakeholders*: project stakeholders.
2.4 *Project motivation model*: project motivations.

3 Product Brief

3.1 *System overview*: overview of proposed solution.
3.2 *High-level product motivation model*: product motivations.
3.3 *High-level role models*: high-level system roles.
3.4 *Assumptions*: product brief assumptions.
3.5 *Constraints*: constraints on the solution.

4 High-level plan

4.1 *Project timeline*: high-level project timeline.
4.2 *Project deliverables*: list of artifacts to be delivered to the client.

5 Endorsement

5.1 *Sign-off*: between the client and the developers.

Fig. 4: A template for the business vision artifact.

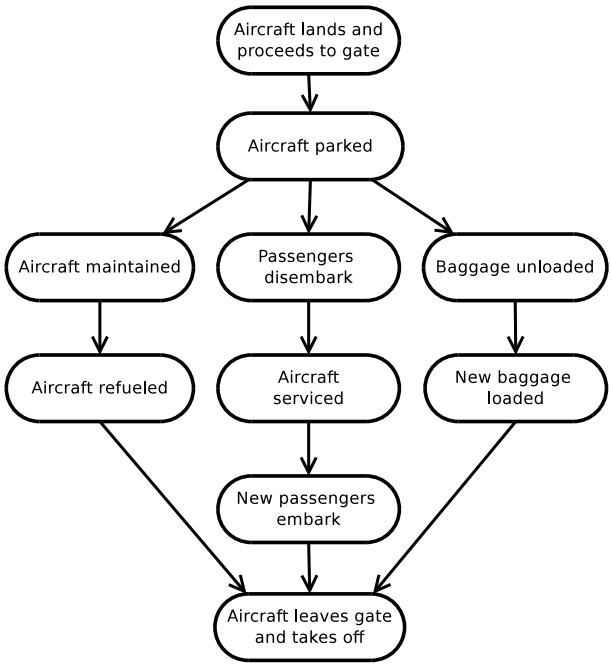

Fig. 5: High-level scenario used to elicit understanding of the aircraft turnaround domain.

system, as any potential solution is likely to have the same motivations. This understanding includes all roles that are part of the system, and the goals achieved, whether these are achieved manually or otherwise.

Our approach uses high-level motivational scenarios of the current system to identify the roles and goals of this system by systematically stepping through the scenarios and answering a series of questions. Motivational scenarios are different to models such as use cases, in that they model interactions between the user and the software system *as well as* interactions that do not cross the boundary of the software system. Scenarios can be derived by the stakeholders, or taken from existing artifacts. Unlike scenario-based requirements techniques [30], our scenarios can be highly unstructured. As a minimum, we require a set of high-level interactions that occur in the system, and dependencies between these.

Figure 5 shows a high-level motivational scenario for the ATS system. Motivational scenarios were provided by the client, and are of a passive nature; that is, there is no discussion of agents/actors themselves.

The approach for eliciting information about the current system is to systematically step through every interaction in the scenarios, one by one, and answer a series of questions about the interactions, recording relevant information in models.

We identify the following questions for eliciting a complete set of models.

Q1 *What is the purpose of this interaction?*
This question aims to elicit the goals and sub-goals of the scenario. Information elicited with this question is recorded in the goal model.

Q2 *Can this interaction be broken into a set/series of smaller interactions?*
This question aims to identify additional goals. If the answer is "yes", add this interaction to the "stack" of interactions to be analysed.

Q3 *Which roles take part in this interaction?*
This question aims to elicit the roles in the system. These roles are recorded on the goal model.

Example: The maintenance of the aircraft is performed to ensure that the aircraft is safe to fly. There are two types of maintenance: routine and non-routine. Routine maintenance is performed by engineers after every flight. Non-routine maintenance is performed by engineers only if requested; for example, by the pilot, to investigate a potential problem. Only the engineers take part in the maintenance of the aircraft, so we add an *Engineer* role to the goal model. In a round-table discussion, we learnt that aircrafts are not re-fueled by engineers, but by *refuelers*. Figure 6 shows the excerpt from the motivation model regarding the aircraft maintenance.

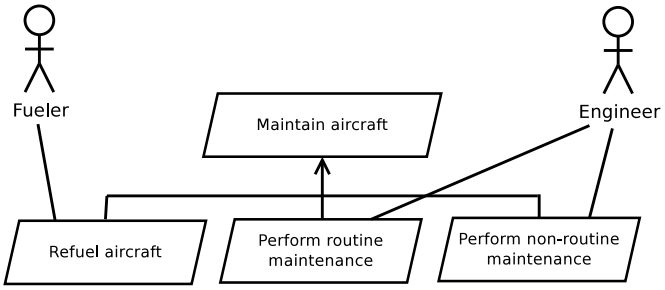

Fig. 6: The sub-goal of maintaining an aircraft during the turnaround process.

Q4 *For each role identified in Question 3*:
Q4.1 *If playing this role, which other roles would I rely*

on, and what are my relationships with these roles?
This question aims to elicit the organisational and interaction relationships of the system. Information elicited with this question is recorded in the organisation model. This question also helps to identify other roles in the system. For additional roles, add them to the queue of roles to be analysed.

Q4.2 *What responsibilities would I have with respect to achieving the goal of this interaction?*
This question aims to elicit the responsibilities of the role. Information elicited with this question is recorded in the *responsibilities* attribute of the role model.

Q4.3 *What knowledge would I require to successfully complete this interaction?*
This question aims to elicit the knowledge required for an agent playing the role to successfully complete the interaction. Information elicited with this question is recorded in the domain model.

Q4.4 *What resources would I require to successfully complete this interaction?*
This question aims to elicit the relevant aspects of the domain/environment. Information elicited with this question is recorded in the domain model.

Q4.5 *To which social policies (rules, regulations, or codes of behaviour) am I required to adhere to successfully complete this interaction?*
This question aims to elicit the constraints under which the role must operate. Information elicited with this question is recorded under the *constraints* attribute of the role model.

Q5 *Are there additional social policies to which participants in the scenario must adhere?*
This question further aims to elicit the constraints under which roles must operate. Information elicited with this question is recorded under the *constraints* attribute of the role model.

Example: The *Engineer* role relies on the *Pilot* role to inform it that non-routine maintenance should be performed, and on the *Manager* role to be instructed to allocate the staff schedule. The *Engineer* is a peer of the *Pilot* role, and is controlled by the *Manager* role.

The *Engineer* role is responsible for undertaking aircraft maintenance, and for this, is required to know the aircraft ID, the gate number at which the aircraft is parked, and that the air-bridge has been positioned. The resources required are the flight plan, staff schedule, and aircraft information. The physical resources are the aircraft itself and the maintenance equipment. Figure 7 shows the role model for this role.

Role ID	R9
Role Name	Engineer
Description	The *Engineer* performs maintenance on the aircraft.
Responsibilities	1. Perform routine maintenance on a specified aircraft when informed of its arrival. 2. Perform non-routine maintenance on a specified aircraft when requested.
Constraints	1. Perform the routine and non-routine maintenance before the scheduled departure of the aircraft.

Fig. 7: The role model for the *Engineer* role.

4.2.3 Eliciting a solution: hiring new staff

To elicit a solution for the problem, we build on the HOMER elicitation technique proposed by Wilmann and Sterling [34], which uses the metaphor of hiring staff in an organisation. The stakeholders are prompted to consider how their problem could be solved by hiring new staff members, perhaps by dropping some of the quality goals, such as "efficiency". For non-technical stakeholders, this metaphor is an intuitive way to conceptualise a solution, and for technically-minded stakeholders, this metaphor forces them to think more about the *"how?"* and *"who?"* aspects of the system.

The questions used to elicit the motivations of the new system are:

Q1 *If one was to hire more staff to handle the problem, what positions would you need to fill?*
This question aims to elicit the new roles that will be added to the system. Information elicited with the question is recorded as roles on the goal model.

Q2 *For each new role identified in Question 1*:

Q2.1 *If playing this role, what is the purpose of my position, and what aspects of the problem would I solve?*
This question aims to define any new goals or subgoals in the system. Information elicited with this question is recorded in the goal model.

Q2.2 Ask Questions 4.1-4.5 from Section 4.2.2.

Q3 *Are there any new social policies to which I must adhere?*
This question aims to elicit any new constraints under which roles must operate. Information elicited with this question is recorded under the *constraints* attribute.

Example: In the turnaround project, new staff could be hired to perform human-based simulations of the turnaround process, allowing evaluation of different resource allocations. Clearly, we were aware that the simulation would be software-based, so we elicited the roles of the system with the knowledge they would be implemented as software agents.

For the purpose of illustration of how this step would work in a non-simulation, we consider an alternate — but closely-related — system, in which we are modelling the ATS in order to add a new re-fueling system to an airport. In this alternative system, an agent fulfilling the *Engineer* role is responsible for re-fueling the aircraft, and there is no *Fueler* role.

The problem that the organisation encounters is that the turnaround is being delayed by the *Engineer* being unable to perform the routine maintenance and refuel the aircraft quickly enough, delaying take off. To solve this problem, the stakeholders identify that they could hire a person to

specifically refuel the aircraft in parallell with the engineer performing routine maintenance.

4.2.4 Defining the solution: deciding the agent types

To define the solution, we must define two things: 1) the software system boundary, which is the boundary between the software and its users and environment; and 2) the behaviour of the software that will solve the problem. We define both of these by specifying the agent types that will play the roles in the system.

Cheng and Atlee [4] advocate defining a system boundary by assigning responsibilities to different parts of the system, such as the software system being constructed, human operators/users, and external systems. We do the same by specifying which roles will be played by human agents, by external systems (either hardware or software), and by software agents.

For example, one possible boundary discussed with the stakeholders in the ATS project was to have the *Manager* role played by a human, instead of a software agent. By changing this assignment of roles, the system changes from a constructive simulator into a human-in-the-loop simulator.

To elicit the behaviour of the system, ask the following questions for each responsibility identified in each role model:

Q1 *If playing this role, what activities would be required for me to fulfil my responsibility?*
This question aims to elicit the behaviour of the system that will fulfil the given responsibility, and contribute to achieving to the system goals.

Q2 *For each activity identified in Question 1*:

- Q2.1 *Is this activity performed by a human agent, an external system, or a software agent?*
  This question aims to assign responsibility to the parts of the system.
- Q2.2 *If performing this activity, what would I have to do?*
  This question aims to break an activity down into sub-activities and atomic actions. Actions are activities that are not considered in further detail. Information elicited with this question is recorded as information about the structure of the corresponding activity in the activity register.
- Q2.3 *What help would I require from other agents to successfully complete this activity?*
  This question aims to elicit possible messages that may be sent and received to complete this activity. Information elicited with this question is recorded in the interaction model.
- Q2.4 *What prompts me to undertake this activity?*
  This question aims to elicit the trigger for the activity; that is, the event to which the agent reacts. Information elicited in this section is recorded as the trigger of the corresponding activity in the activity register. If this trigger is a message received from another agent playing some role, it is also recorded in the interaction model.
- Q2.5 *Under what conditions can I undertake this activity?*
  This question aims to elicit the precondition for the activity; that is, the states of the environment that enable this activity. Information elicited with this question is recorded as the precondition of the corresponding activity in the activity register.
- Q2.6 *What happens after I complete this activity?*
  This question aims to elicit how the activity changes the environment. Information elicited in this section is recorded as the postcondition of the activity.
- Q2.7 *What other agents to I need to inform that this activity has been completed?*
  This question aims to elicit information regarding interactions, and the actions for this activity. This information is recorded in two places: 1) as a part of high-level description of the actions for this activity (as in Question 2.2); and 2) in the interaction models.

Example: In the ATS system, the routine maintenance will be performed by a software agent, and is modelled as an atomic action for simulation purposes. To perform routine maintenance, the aircraft must have its wheel chocks positioned, and the activity is triggered when the *Engineer* is informed by *Airport Ground Staff* that the aircraft is ready for maintenance. After the routine maintenance activity is complete, the aircraft is in a state in which the *Engineer* deems that it is safe to fly, and it informs the *Pilot* of this. The final activity description for the routine maintenance is shown in Figure 8.

Activity name:	Routine maintenance.
Trigger:	Informed by *Airport Ground Staff* of the aircraft ID of the aircraft that is ready for maintenance.
Precondition:	Wheel chocks of the aircraft ID are in position.
Sub-activities:	1. Perform the routine maintenance on the specified aircraft. 2. Inform *Pilot* that the routine maintenance is complete on the aircraft.
Postcondition:	Aircraft with the specified ID is safe to fly.

Fig. 8: Activity description for *Routine maintenance* activity.

The final step is to decide how the agent types will be defined to play the roles, and to perform the activities. We do not define a solution for this in this paper, as there are a number of useful methodologies that define this [17], [25], [37], [28]. In our experience, agent types can be defined as a one-to-one mapping for simulation systems; that is, each role is mapped to one agent type, with the activities relating to a responsibility also mapped to the agent. In the ATS system, we used a one-to-one mapping, which was intuitive and clear from the role definitions, and is likely to be similar for any simulation system. However, there will be cases in which a one-to-one mapping is not possible. One clear case is when the responsibilities defined by a single role are mapped to multiple activities, but the activities are split over different categories; e.g. some are identified as being played by a human agent,

and some by a software agent.

An an example of an agent type, consider the *Engineer* agent, which fulfils the role of *Engineer*. The analysis of the activities results in the definition found in Figure 9.

Name:	Engineer
Description:	Play the role of *Engineer* by performing routine and non-routine aircraft maintenance.
Activities:	**Activity name:** Routine maintenance ... **Activity name:** Non-routine maintenance ...
Environment considerations:	1. Aircraft 2. Aircraft information 3. Flight schedule 4. Aircraft gate number 5. Staff schedule

Fig. 9: Agent type specification for the *Engineer* agent.

The end result of the process is a set of agent types, which fulfil the system roles, and a description of the behaviour of these agents, specifying what the agent does, and how this affects the environment.

5 Specification and Packaging

The authors are unaware of any agent-oriented methodologies that provide support for creating deliverables such as software requirements specifications, even though these deliverables are a vital part of many software engineering projects. In this section, we present the definition of a *software requirements specification* (SRS) — a deliverable describing the software system to be built. An SRS defines a shared understanding between the project stakeholders as to what is to be built; thus, it is an agreement between these stakeholders.

5.1 Describing socio-technical systems using agent models

Using the organisational metaphor to elicit the requirements of a socio-technical system is beneficial, and we extend this to the specification of the system as well.

To specify the system, we adopt the systems theory view, acknowledging that organisations and socio-technical systems are systems in their own right. The view of a system is broken in structure, behaviour, interaction, and purpose. The purpose of a system influences its structure and behaviour, which also influence each other, and the interactions. To understand a system, one cannot view any of these aspects in isolation; they must be considered as a whole. Figure 10 shows where Sterling and Taveter's models reside with regarding to the systems view.

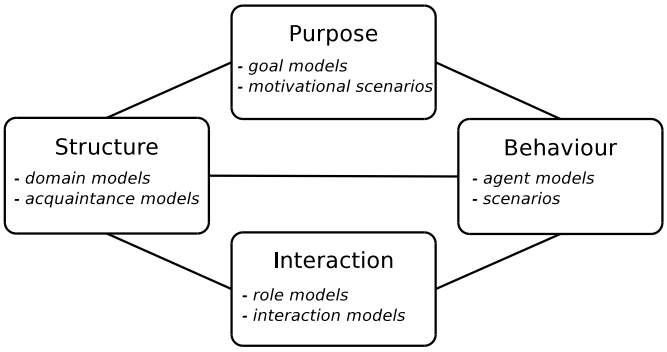

Fig. 10: The structure of socio-technical systems, as viewed from the agent paradigm.

Many agent methodologies consider the definition of agents as a design step (e.g. see [37, Fig. 6]), and hold the view that including this in the specification unnecessarily constrains design (see [28, Ch.6]). Using the models described in Section 3, this would imply that a specification consists of role models, organisational models, domain models, goal models, and motivational scenarios. It is our view that these models do not provide the "what" required to produce an unambiguous software requirements specification.

We advocate the inclusion of (at least some of) the models in the platform-independent computational design [28]. Those models that we deem necessary to include are the *agent models*, which determine which agents play each role, and the *agent behaviour models*, which determine the behaviours of the agent types in terms of the activities that they perform. Additionally, we believe that it would prove valuable to include the interaction models, which determine the protocols between agents, and the knowledge models that are used for representing the agents' knowledge. Other agent-oriented methodologies could be interpreted in a similar manner.

5.2 Packaging the SRS

Producing a complete SRS requires us to package these models together in a meaningful way. Figure 11 presents a possible template for an agent-oriented SRS, based on existing templates such as Wiegers' SRS template [33] and the IEEE Standard for requirements specifications [16]. 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.

One can see from Figure 11 that the SRS is not structured the same as Figure 10. Instead, the specification is presented based on abstraction, with higher abstraction being presented earlier. The scenarios in Section 7 of the template refer to those scenarios at the platform-independent design layer, so are presented later rather than earlier.

Sections 1 and 2 of the template are similar to that of Wiegers' [33], except that Section 2 includes the high-level motivational model (e.g. Figure 3), and Section 2.4 of Wiegers' template (Operating Environment) has been considered as

Title information

Revision history

Table of contents

Fig. 11: A software requirements specification template based on the elicited models.

its own section (Section 5), emphasising the importance of the environment in socio-technical systems. Wiegers' *System Features* section has been replaced by the agent types and interaction models (Section 6), as this defines the behaviour of the system. Sections 3 and 4 in our template have no equivalent in Wiegers' template, as Wieger does not consider the purpose of the system and its roles. Wiegers' *Other Nonfunctional Requirements* section is not included, as nonfunctional requirements are considered as quality goals in the goal models.

5.2.1 A note on external interfaces

Section 8 of the template in Figure 11 advocates specifications of the external interfaces of the system. To identify the interactions that are involved in each of these interfaces, we consider the organisational model and the agent types.

Recall that in our approach, agents are classified as human agents, software agents, or external hardware/software systems. We identified that any *new* software agents would be part of the software system that is to be implemented. Therefore, by analysing the interactions between these new agents and the other agents in the system, we can determine where the external interfaces will be, and use this to feed into the interface specification.

The categorisation of the agent types provides a clear divide for each interface type:

- interactions between a software agent and a human agent will take place at the user interface;
- interactions between a software agent and an external hardware or software system will take place at the hardware or software interface respectively; and
- interactions between a human agent and an external hardware/software agents fall outside of the software boundary, so are not considered.

6 Verification and Validation

In this section, we discuss the verification and validation (V&V) of the requirements specification, specifically the models in the specification. V&V was performed iteratively throughout the process. In the ATS project, three main methods for V&V were applied were technical reviews, prototyping, and cross-validation of models.

6.1 Reviewing and prototyping

Reviewing and prototyping are common methods for V&V of software systems. In previous work, two of the authors have explored the use of prototyping using the models [29], [31]. This paper does not introduce any new techniques in reviewing and prototyping. Nonetheless, we emphasise the importance of these in our requirements engineering process, and also include this as completeness in the case study.

Technical reviews were undertaken both by individual members of our team, as well as during round-table meetings with our industry partners. In the round-table meetings, the models were discussed and scrutinised, and any omissions or corrections were made in the meeting, thus implying an implicit informal review.

The means of prototyping the system was useful for locating problems. A prototype of the system was produced by an undergraduate software engineering student at the University of Melbourne, directly from the SRS. The student implemented

the prototype as a multi-agent system using the AgentSpeak language and the Jason interpreter [2].

The student's feedback into the models was important for us as he had not been a part of the project beforehand, so therefore he gave an external viewpoint. It is interesting to note that the student commented on the ease of using the SRS. Although he was provided with several references on understanding agent systems and a textbook on Sterling and Taveter's models [28], the student commented that these were largely unnecessary because the agent-oriented models were straightforward to interpret. The prototype itself was used as a validation tool. Over two meetings, the scenarios from the SRS were walked through with our industry partners, who highlighted several incorrect assumptions that we had made, and also highlighted some key requirements that were missing from the SRS. As a result of the prototyping, we were able to confirm that the final SRS is correct and complete with respect to our partner's needs.

6.2 Cross-validation of models

Our contribution to V&V is in the form of a systematic cross-validation method. The validation consists of checking the consistency between the models and an ontology that models the problem as conceptualised by the client. The ontology models the important concepts and relationships between the concepts in the system. Any inconsistency between the two indicates a problem with either the ontology or the models. The process used is described by Lopez et al. [19]. In this section, we overview this and discuss some of the results.

The ontology was developed based on documentation that the client provided as well as several interviews with them. The team deriving the ontology worked separately from the other members in the requirements engineering phase to introduce diversity between the models and the ontology. Throughout the development, the ontology is revisited to ensure that it is updated with any additional insights the client develops through interacting with the development team. The ontology consists of 350 concepts and relations, most of which were identified from documentation provided by the client. In addition, the ontology is augmented with annotations describing concepts and relations specifically relevant to agent-oriented models. Table 1 illustrates these properties for the models at the conceptual domain modelling level. These properties provide software engineers with a straightforward way to evaluate the consistency between the ontology and the other models.

The validation activity is iterative: models are validated as soon as they are available, correcting errors as they arise and avoiding compounding and propagating errors to later phases of the development.

The validation process consists of applying, for every model developed, a list of model-dependent operators. Applying an operator can trigger one or two proposals to amend the model, depending on the outcome. For example, validating the agent-oriented model consists of ten operators that can trigger amendment proposals, exemplified by the following: "*add to the model set an agent type* X", where X is defined in the ontology but it does not have a corresponding model. To ensure

TABLE 1: Ontology Properties

Domain	Property	Range
Goal	*is a*	Goal
Goal	*part of*	Goal
Role	*responsible for*	Goal
Role	*participates in*	Activity
Role	*is peer*	Role
Role	*controls*	Role
Role	*is benevolent*	Role
Role	*uses*	Environment
Agent	*plays*	Role
Agent	*performs*	Activity
Activity	*fulfils*	Goal
Activity	*requires*	Environment
Activity	*precedes*	Activity
Activity	*follows*	Activity

effectiveness of the cross-validation activity, the creators of the models were not directly involved in the validation. Instead, as the requirements engineering was undertaken, the modellers received recommendations from the team members undertaking the cross validation. Iterations were undertaken until models converged and no further amendments were proposed by the validation activity.

The initial set of models that underwent validation included environment, goal, role, organisation, interaction and scenario models. During subsequent iterations, further models were added (e.g. an agent model). Some models that were developed in close consultation with the client converged quickly, as they were close to what was expected and the client had detected discrepancies beforehand. In particular, the goal model and the role model underwent minor improvements and were stable after the second iteration. Other models were changed throughout four iterations of the activity, including the domain model, the organisation model, the interaction model, the agent model and the scenario model. Additional details can be found in earlier work [19]. The cross validation against the ontology highlighted the following opportunities to improve the models:

- some missing relations were detected (e.g. in the environment model);
- discrepancies in models were detected (e.g. in the roles relationships within the organisation model and in the messages proposed in the interaction model); and
- scope for further refinement was highlighted (e.g. parallel processing opportunities were identified in the interaction model, more details were proposed in the scenarios).

Following the third iteration, the number of proposals produced by the validation process had largely converged.

7 Lessons Learnt

In this section, we discuss the most important lessons that we learnt as part of the project, about our method and about agent-oriented requirements engineering in general.

7.1 Requirements elicitation

Business vision documents. Business vision documents are typically not required for academic projects, and therefore

have not been addressed in the agent literature as far as the authors are aware. The use of high-level models in such documents is a novel application of agent-oriented models that was significant for our industry partner. The business vision models were helpful in identifying the high-level motivations and the stakeholders. For example, within seconds of being presented the first draft of the model in Figure 3, one of the industry partners noted that the air traffic controllers were stakeholders in the turnaround process, and this induced discussion about how new traffic enters the airport. In subsequent iterations, the air traffic controller role was deemed unnecessary for the simulation and was dropped, but changes related to this remained. It is not unreasonable to claim that if the scope of the system was larger, we would have been required to engage with air traffic controllers as part of the elicitation process, and we believe our model would have identified this earlier than otherwise. Events such as this further strengthen our view that using stakeholders as modellers is highly valuable.

Elicitation questions. The elicitation for the ATS system did not follow the questions in the order listed in Section 4.2, and we expect that this would be the case for other projects. The elicitation questions form a checklist, but the order in which they are asked did not seem important. Conversations were triggered by stakeholders, and we found it important to allow these conversations to occur and to record the details for further analysis, rather than fixating on the questions.

However, the questions did form the basis of some conversations, and our industry partners found this approach effective for bringing themselves into the agent mindset; something with which they had struggled previously.

Stakeholders as modellers. A key characteristic of our elicitation approach is the use of stakeholders as modellers. In our elicitation meetings, each stakeholder was provided with a copies of the most recent models, and comments were invited. During the meetings, these models were modified by the group, thus taking advantage of the experts' knowledge of the domain. We found it necessary for the requirements team to perform further analysis in between meetings to ensure consistency between different models, etc., and to refine lower-level models.

We found that the lightweight nature of the models was useful in the meetings, as many incorrect assumptions that we had made were quickly identified by the domain experts, and corrected. This is consistent with our previous work with less-technical stakeholders.

Abstraction in understanding. The lower-level and less graphical models, such as the agent type specifications, were less useful in meetings, due to the inability to consider many of these models at one time. The motivational models (role models, goal models, and domain models), were more useful, even at a high level. We found that lower-level models were largely produced outside of the round-table meetings. This is especially the case for the agent types. We believe that defining agent types outside of these meetings would be more straightforward than defining motivation models, due to having a better understanding of the system by this time.

The usefulness of high-level models is evident from an example. Early in the process, one of the requirements engineers devised a high-level domain model, which contained the concepts he thought were relevant, and links between these concepts. The links represented relationships, but did not define what these relationships were, as the engineer had not yet identified these. Initially, these were met with confusion from the other members in the meeting. The value of this model became clear only minutes later when one of the domain experts identified several incorrect assumptions about the relationships between concepts in the model, despite not knowing what the relationships meant. This indicates that, at least early in the requirements process, it is beneficial to share *any* understanding of the system, rather than waiting until models are complete.

Software system boundaries. We advocate delaying the definition of the system boundary until as late as is reasonable/possible, and at least until stakeholders have a shared understanding of the problem. Project or organisational constraints may require a system boundary early in the project, perhaps before the problem is fully understood. In these cases, we believe that the boundary decision should be delayed as long as possible without affecting the remainder of the project; e.g. contract agreements.

The software system boundary was left undefined for most of the requirements elicitation process. Our industry partners did not feel that it impacted the project negatively, however, in this particular case study, they did not see any benefit in delaying the boundary definition either, because they felt that the only obvious boundary was one in which all roles were played by software agents, although they did see that this could be useful for other systems.

To our group, the benefits of not defining a system boundary at the start of elicitation are illustrated by the project. We described earlier in this section the discussion that was held regarding whether the role of the *Manager* was to be played by a human or a software agent. Had the boundary been defined at the start of the requirements elicitation, this discussion may not have taken place. It is examples such as these that strengthen our claim that delaying definition of software system boundary can be beneficial.

The interaction designers we have collaborated with in other work [24], [27] have embraced the idea of delaying the software system boundary. A colleague (interaction designer) reported to us his experience with designing technology to support school children with autism. In working with a group of school teachers who were specialists in teaching autistic children, he found that by not constraining the system boundary, the teachers produced more useful solutions. By simply asking the teacher groups to design a technologically-based solution to support the children, the groups attempted to fit everything into software. The teacher groups who did not have this mandate all included technology as part of their designs, but the support was extended well outside of the software system boundary.

This is consistent with Gause's view [12], which states that taking the time in early requirements engineering to discuss possible solution boundaries with stakeholders raises awareness about possible solutions, and can discover *deep context regions* — those areas factors that are often oversights

until a product is released.

Model evolution. As expected, our experience indicates that having models evolve over series of round-table discussions leads to a clearer solution, as early concerns regarding concepts such as resources were delayed without jumping to a pre-conceived solution. Later in the development process, successive versions of the models enabled traceability of the design decisions that were made, and of the requirements in general. This gave the research team something to fall back on when discussions started to get too complex for some stakeholders or drifted from original high-level goals, and also made the source of requirements more straightforward to trace. 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 [20].

7.2 Requirements specification and packaging

Agent types. The major lesson that we learnt as part of this project was with regards to the inclusion of agent types in the SRS. In previous work, we had, like other researchers, considered agent types as a design artifact. However, during this project, we came to the conclusion that the use of agent types to define behaviour is important.

Early in the ATS requirements process, our collaborators struggled to identify the behaviour of the system, or how they could implement and verify a system against a set of high-level models. However, once we decided to include agent types, the system behaviour became much clearer to them.

We believe that defining the behaviour is important with regards to obtaining a sign-off from the client. The signing off of requirements, and what constitutes this sign off, is overlooked in academic research on agent-oriented software engineering, but is important for contract definition in projects with third-party vendors.

Our collaborators at Jeppesen particularly like the flexibility enabled by the agent paradigm and the use of agent types. In their experience, clients on different projects are often happy to sign-off at different levels of abstraction. For example, some clients would be happy to sign-off the role and goal models, while others would want to see the more detailed agent types. In an event-based system, this distinction is less clear.

SRS Template. For the ATS project, the SRS of the system closely follows the structure recommended in this section, with some changes to suit the specific system. As part of the ATS specification, interaction models were derived, however, they were omitted from the latter versions of the SRS because we felt they did little to help define or understand the system behaviour. The other stakeholders agreed that the interaction models gave them little value in understanding the proposed solution. This is largely because the interaction protocols used in the ATS system were either largely sequential or were straightforward enough to extrapolate the interactions from the agent types and scenarios. The interaction diagrams were included in the software design.

In our view, the final SRS for the project is a well-packaged artifact. We believe the SRS to be correct, complete, and consistent, a view that is strengthened by our industry partners, who have endorsed (signed-off) the SRS package. This sign off is an agreement between ourselves and our industry partners that the requirements are correct, complete, and consistent, showing that our approach can be used to arrive at a solution with which all stakeholders are satisfied. We see this as an important result in itself.

8 Related Work

Agent-oriented requirements engineering has been investigated by other authors, and as a result, several methodologies have been proposed, such as Tropos [3], Prometheus [25], Gaia [37], INGENIAS [26], and ROADMAP [17]. Blanes et al. [1] performed a systematic of agent-oriented requirements engineering, finding that most research in the area focused on notations for modelling and analysis, with little support for requirements elicitation, specification (other than modelling), or validation.

8.1 Requirements elicitation and analysis

Both agent-oriented and goal-oriented requirements elicitation and specification have been investigated in the past. A key feature of much of the existing work is on motivations; that is, the "whys" of a system, in addition to the "whats". Our approach continues in this direction, and we have found this to be valuable in understanding systems.

Two major differences between our work and other agent-oriented and goal-oriented requirements elicitation methods are in the level of detail. First, similar to NFR [22], [5] and other works e.g [13], we acknowledge that committing to a design decision too early may result in some stakeholders' solutions being discarded, making their views irrelevant. However, a key contribution of this work compared with other work is that it further encourages stakeholder involvement by facilitating the inclusion of all key stakeholders in the *modelling* and *analysis* of the system, not just the elicitation. Zowghi and Coulin [39] note that, especially in group meetings, stakeholders must feel confident that their views will be heard, and that they are part of the process. We encourage stakeholders to discuss *and modify* models during group meetings help to engage them in the requirements process. In other work, we have successfully employed agent-oriented models in this context with psychologists, ethnographers, and interaction designers [24], [27]. We believe that our approach could be applied to other agent- or goal-based modelling notations by using just a simple subset of the notations with which non-technical stakeholders would be comfortable.

Second, we prescribe a more detailed approach for eliciting information and recording it into models. The KAOS methodology includes a requirements acquisition technique [6] similar to ours. KAOS identifies what is required for the final models (system goals, agents, action, and domain attributes), but is less prescriptive in how to arrive at these. The Tropos methodology [3] uses a question-answer technique for eliciting requirements. The Tropos requirements elicitation technique involves four questions: *who are the main actors?*; *what are their goals?*; *how can they achieve them?*; and *does an actor*

depend on another to achieve its goals?. These questions are broader versions of our questions for understanding the current system, and do not define how to arrive at solutions.

The social organisation metaphor has been used to analyse and specify requirements. Donzelli and Bresciani [8] use goal modelling to develop, during the analysis phase, an organisational view of agent-oriented systems. Yu [35] stresses the importance of identifying motivations within an organisational context in early-phase requirements engineering. Yu proposed the i^* modelling language to capture these motivations, commenting that social considerations are not commonplace among most modelling techniques. Yu's notion that software processes can be modelled as social processes is the essence of using agents to implement roles in our work.

Many of these concepts are inherited by the Tropos agent-oriented development methodology [3], which is built on i^*. Blanes et al. identify Tropos [3] as providing the most mature support for requirements elicitation. Like Tropos, our approach asks "why" as well as "what" when eliciting the requirements, because we agree that the motivation of the system is important for understanding how the system will fit within its organisation and environment. However, this research has not been translated into a standardised method to explicitly elicit requirements for agents using organisations as the guiding metaphor. Our approach provides a systematic and repeatable approach for eliciting requirements, which we believe could be used within the Tropos methodology.

8.2 Requirements specification

A major difference between our work and other agent-oriented requirements engineering methods is the inclusion of agent types in the requirements specification. Typically, this is considered as design restriction, and therefore not good requirements engineering practice. However, the purpose of requirements engineering is to define the external behaviour of the system in its environment. Role specifications assign responsibilities for achieved goals, but purposely omit definitions of behaviour. Thus, multiple systems, each with different behaviour from the others, could achieve the specified goals.

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. [10] identify two approaches for specifying behaviour of a multi-agent system: assigning individual requirements to individual agents; and assigning behaviour to *role instances*, which are further refined into agents. The first specified behaviour from the perspective of an external observer, while the second specifies behaviour from the viewpoint of the individual instance, which is closer to our approach; however, we feel that the intermediate representation between roles and agents is unnecessary, and that our approach of assigning responsibilities to agents is a cleaner solution.

KAOS [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. KAOS does not distinguish between roles and agents, instead treating agents as the primary actors that achieve goals. The constraints related to goals are assigned to agents using *responsibility links*, making their notion of an agent a merging of our notation of roles and agents. When applying KAOS, van Lamsweerde et al. [32] comment that the last stages of the *goal elaboration* process "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 [25], like KAOS, does not consider roles as part of requirements engineering. Like 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* are natural language descriptions of behaviour. Prometheus has been applied to industry applications, such as the meteorological alerting system developed with the Australian Bureau of Meteorology [21].

We have not seen other work that outlines how a requirements package should be constructed for a multi-agent system. Several other generic SRS package templates exist, such as that from Wiegers [33] and the IEEE Standard for requirements specifications [16]. These two templates are both similar to our template, however, we see the value in using a template that emphasises the concepts that are central to the agent-oriented paradigm.

9 Conclusions

In this paper, we described two improvements to previous work on agent-oriented requirements engineering. These improvements relate to problems experienced by our industry partner: (1) a lack of systematic methods for agent-oriented requirements elicitation and modelling; and (2) a lack of prescribed deliverables for agent-oriented requirements.

Our elicitation approach prescribes a list of questions to be answered by stakeholders in round-table meetings, and how to directly map the answers to lightweight agent-oriented models. Further, we prescribe a requirements specification template that uses agent-oriented models as a central focus. Importantly, the template advocates the inclusion of agent types at the requirements level, rather than defining these at design time, based on our observation that roles, goals, and interactions alone are not sufficient for describing system behaviour.

We have been fortunate enough to attract an industry partner to work closely with to improve our requirements engineering processes — something that is difficult to do for researchers in this area. Applying our approach in conjunction with our industry partner demonstrated that the approach is useful, and we believe led to a much better requirements engineering method. A strong result of this work is that our industry partner has adopted many parts of our requirements engineering method into their own requirements engineering process models. This validates our claims that using the agent paradigm is not merely an academic exercise.

Current and future work will explore our requirements engineering approach in more detail. Currently, three individuals of varying experience are designing and implementing different versions of the ATS system from our requirements specification. We will assess the differences and similarities of

these systems to evaluate which parts of our approach need to be improved. Ideally, we will see three systems that display the same behaviour. In addition, we are designing, implementing, and testing our own version of the ATS system to evaluate Sterling and Taveter's design models [28].

In future work, we will apply our approach to larger-scale systems where the domain requires more detailed analysis, with the aim of evaluating how our approach scales. Already our industry partner is applying their modified requirements engineering process model to the maintenance of large-scale air-traffic simulators. This domain is highly complex, so we expect to receive useful feedback.

Acknowledgements

This research is funded by the Australian Research Council Linkage Grant LP0882140.

REFERENCES

[1] D. Blanes, E. Insfran, and S. Abrahão. Requirements engineering in the development of multi-agent systems: a systematic review. In *Intelligent Data Engineering and Automated Learning*, pages 510–517. Springer, 2009.

[2] R. H. Bordini, J. F. Hübner, and M. J. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. Wiley-Interscience, 2007.

[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] 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, Workshop on the Future of Software Engineering*, pages 285–303, 2007.

[5] L. Chung and J.C.P. Leite. On non-functional requirements in software engineering. In *Conceptual Modeling: Foundations and Applications*, volume 5600 of *LNCS*, pages 363–379, 2009.

[6] A. Dardenne, A. Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1-2):3–50, 1993.

[7] S.A. DeLoach, M.F. Wood, and C.H. Sparkman. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

[8] P. Donzelli and P. Bresciani. Improving requirements engineering by quality modelling — a quality-based requirements engineering framework. *Journal of research and Practice in Information Technology*, 36(4):277–294, 2004.

[9] K. El Emam and A.G. Koru. A replicated survey of IT software project failures. *IEEE software*, 25(5):84–90, 2008.

[10] J. Ferber, O. Gutknecht, C.M. Jonker, J.P. Müller, and J. Treur. Organization models and behavioural requirements specification for multi-agent systems. In Y. Demazeau and F. Garijo, editors, *Proceedings of the 10th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 1–19, 2001.

[11] 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.

[12] D.C. Gause. Why context matters-and what can we do about it? *Software, IEEE*, 22(5):13–15, 2005.

[13] V. Gervasi and D. Zowghi. On the role of ambiguity in RE. In *Requirements Engineering: Foundation for Software Quality*, volume 6182 of *LNCS*, pages 248–254, 2010.

[14] Standish Group. Chaos report, 1994.

[15] R. Guizzardi and A. Perini. Analyzing requirements of knowledge management systems with the support of agent organizations. *Journal of the Brazilian Computer Society*, 11(1):51–62, 2005.

[16] IEEE. IEEE Std 830-1993, Recommended practice for software requirements specifications, 1998.

[17] T. Juan, A. Pearce, and L. Sterling. ROADMAP: Extending the Gaia methodology for complex open systems. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 3–10. ACM Press, 2002.

[18] 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.

[19] D. A. Lopez, G. Beydoun, L. Sterling, and T. Miller. An ontology-mediated validation process of software models. In *19th Int. Conf. on Information Systems Development*, pages 455–467. Springer, 2011.

[20] 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 IFIP TC13 Third International Conference on Human-Computer Interaction*, pages 207–212, 1990.

[21] I. Mathieson, S. Dance, L. Padgham, M. Gorman, and M. Winikoff. An open meteorological alerting system: Issues and solutions. In *Proceedings of the 27th Australasian Conference on Computer Science*, volume 26, pages 351–358. ACS, 2004.

[22] R. Mehta, H. Wang, and L. Chung. Dealing with NFRs for smart-phone applications: A goal-oriented approach. *Software Engineering Research, Management and Applications*, pages 113–125, 2012.

[23] T. Miller, S. Pedell, L. Sterling, and B. Lu. Engaging stakeholders with agent-oriented requirements modelling. In *Agent-oriented Software Engineering XI*, volume 6788 of *LNCS*, pages 62–78, 2011.

[24] 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.

[25] L. Padgham and M. Winikoff. *Developing Intelligent Agent Systems: A practical guide*. John Wiley and Sons, August 2004.

[26] 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.

[27] S. Pedell, T. Miller, F. Vetere, L. Sterling, S. Howard, and J. Paay. Having fun at home: interleaving fieldwork and goal models. In *Proceedings of OZCHI*, pages 309–312, 2009.

[28] L. Sterling and K. Taveter. *The Art of Agent-Oriented Modeling*. MIT Press, 2009.

[29] L. Sterling and K. Taveter. Event-based optimization of air-to-air business processes. In N. Stojanovic, A. Abecker, O. Etzion, and A. Paschke, editors, *Proceedings of the Intelligent Event Processing – AAAI Spring Symposium*, pages 80–85. AAAI Press, 2009.

[30] A. Sutcliffe. Scenario-based requirements engineering. In *Proceedings of the 11th IEEE International Requirements Engineering Conference*, pages 320–329. IEEE, 2003.

[31] K. Taveter and L. Sterling. An expressway from agent-oriented models to prototype systems. In *Agent-Oriented Software Engineering VIII*, volume 4951 of *LNCS*, pages 192–206, 2007.

[32] A. Van Lamsweerde, R. Darimont, and P. Massonet. Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. In *2nd IEEE International Symposium on Requirements Engineering*, pages 194–203. IEEE Computer Society, 1995.

[33] K. Wiegers. *Software requirements*. Microsoft Press, 2nd edition, 2003.

[34] D. Wilmann and L. Sterling. Guiding agent-oriented requirements elicitation: HOMER. In *Fifth International Conference on Quality Software (QSIC)*, pages 419–424, 2005.

[35] E. Yu. Social modeling and i^*. *Conceptual Modeling: Foundations and Applications*, pages 99–121, 2009.

[36] F. Zambonelli, N. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 231–251. Springer, 2001.

[37] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multi-agent systems: The Gaia methodology. *ACM Transactions on Software Engineering Methodology*, 12(3):317–370, 2003.

[38] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering Methodology*, 6(1):30, 1997.

[39] D. Zowghi and C. Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. In A. Aurum and C. Wohlin, editors, *Engineering and managing software requirements*, chapter 2, pages 19–46. 2005.