A Thesis on

Management Relationships in
Self-managed Service Compositions

Justin King

Submitted for the degree of Bachelor of Science (Computer Science) (Honours)

Faculty of Information and Communication Technologies
Swinburne University of Technology

June 2009
Abstract

Self-managed systems have been identified as a key approach to addressing the problem of ever increasing software complexity. Systems with self-management capabilities decrease the burden on human operators, as issues such as problem diagnosis, configuration and responses to changing non functional requirements are taken on as an internal responsibility. Building self-managed service compositions is one way of embedding a system with self-management capabilities. These systems take existing web services and other components with no self-management capability and introduce an autonomous entity responsible for monitoring the environment and adapting the service configuration to keep operation optimal.

Systems that possess this level of autonomy are often functionally decomposed into organisational hierarchies. These hierarchies are strict command-control structures, yet as systems become more open and exist across domains there is a need to allow control relationships to exist independently of structure and for more flexible structures to be possible. This will allow more effective use of the increasing intelligent capabilities of autonomous systems. Management patterns in human organisations allow these flexible structures; therefore it is desirable to attempt to allow software systems to be structured using similar patterns.

The research undertaken in this thesis seeks to further the ability to structure self-managed systems into flexible structures normally found in human organisations. Several existing approaches exist that allow modelling of management relationships in self-managed systems. ROAD is a framework that defines service compositions as adaptive, self-managed structures and has been selected to serve as a base for the work in this thesis.

The contribution of the thesis is as follows. A management communication model is defined, which categorises systems into different levels of management capability and explores the messaging requirements of each level. Furthermore a management interface is defined for a web services environment, which is designed to cater for these requirements and to help facilitate systems with differing capabilities to establish complex management relationships. Additional constructs are also defined which further the ability of ROAD to structure systems according to management patterns found in human organisations. Finally a prototype implementation extending the ROAD framework is developed which demonstrates the feasibility of the concepts described in this thesis.
Acknowledgements

I would like to sincerely thank my supervisor Dr Alan Colman for taking me on as a research student during the last semester of my undergraduate degree, as well as my honours year. Thanks Alan for the help with scholarships, the much needed guidance, as well as the time spent giving feedback on drafts and ideas. My thanks also go to Professor Jun Han and Dr Minh Tran for their input on various parts of my work, as well as fellow research student Malinda Kaushalye Kapuruge for his contribution to the ROADfactory project.

Thanks to my Mum and Dad for all the help and support, financial and otherwise, that has allowed me to spend the last 5 years studying. In particular for the helping me in the early stages to get started in Melbourne, and for moving me to Adelaide and then back again 12 months later for my placement. You have helped me out far more than you had too and I’ve tried my best to make your efforts worthwhile.

Last but not least thanks to my girlfriend and proof reader Jasmine for putting up with me during the stressful times and for teaching me you don’t need to put comas after the words ‘and’ and ‘or’.
Declaration

This is to certify this thesis contains no material which has been accepted for the award of any other degree or diploma. To the best of my knowledge, this thesis contains no material previously published or written by other people except where due reference is made.

Justin King
June 2009
Table of Content

1 INTRODUCTION ............................................................................................................ 1

1.1 RESEARCH QUESTIONS ............................................................................................ 2
1.2 THE APPROACH TAKEN ......................................................................................... 2
1.3 LIMITATIONS AND DELIMITATIONS ..................................................................... 3
1.4 RESEARCH CONTRIBUTION .................................................................................... 3
1.5 THESIS OUTLINE .................................................................................................... 4
1.6 SUMMARY ............................................................................................................... 5

2 BACKGROUND ............................................................................................................... 6

2.1 MANAGEMENT AND ORGANISATIONAL STRUCTURES ........................................ 6
2.2 TOWARDS MANAGEMENT STRUCTURE IN SOFTWARE SYSTEMS .................. 10
2.3 MANAGEMENT IN EXISTING SOFTWARE SYSTEMS .......................................... 13
   2.3.1 Simple Network Management Protocol ......................................................... 15
   2.3.2 Common Information Model / Web-Based Enterprise Management ............. 17
   2.3.3 Web Services Distributed Management ....................................................... 18
   2.3.4 Web Services for Management ..................................................................... 19
   2.3.5 Java Management Extensions ....................................................................... 20
   2.3.6 Web Service Offerings Language ................................................................. 21
   2.3.7 Evaluation ...................................................................................................... 21
2.4 POLICY BASED MANAGEMENT .............................................................................. 22
   2.4.1 WS-Policy ...................................................................................................... 22
   2.4.2 Ponder .......................................................................................................... 23
   2.4.3 Rei ................................................................................................................ 24
   2.4.4 Evaluation ...................................................................................................... 25
2.5 ARCHITECTURAL FRAMEWORKS SUPPORTING SOFTWARE MANAGEMENT STRUCTURES .............................................................................................................. 25
   2.5.1 Lightweight Architecture for Reconfiguring Applications ............................ 25
   2.5.2 Viable Systems Architecture ........................................................................ 26
   2.5.3 Evaluation ...................................................................................................... 28
2.6 SUMMARY ............................................................................................................... 28

3 A MOTIVATING EXAMPLE ...................................................................................... 30
List of Figures

FIGURE 2.1: THE FIVE PATTERNS OF MANAGEMENT (MINTZBERG, 1983) .......... 8

FIGURE 2.2: ILLUSTRATION OF THE CORE PARTS OF AN ORGANISATION AND
MANAGEMENT PATTERNS ......................................................................................... 9

FIGURE 2.3: THE THREE STANDARD MANAGEMENT PROTOCOL
COMMUNICATION PATTERNS ..................................................................................... 15

FIGURE 2.4: AN OVERVIEW OF A SNMP ENABLED NETWORK ............................ 16

FIGURE 2.5: JMX ARCHITECTURE OVERVIEW .................................................... 20

FIGURE 2.6: AN EXAMPLE LIRA AGENT HIERARCHY (CASTALDI ET AL., 2003) 26

FIGURE 2.7: A SYSTEM DESIGNED USING THE VSA ARCHITECTURE (HERRING,
2002) ......................................................................................................................... 27

FIGURE 4.1: AN EXAMPLE ROAD SMC ILLUSTRATING THE STOCK EXCHANGE
........................................................................................................................................ 36

FIGURE 4.2: THE AUTOMOTIVE EXAMPLE FROM CHAPTER 3 AS ROAD SMC’S 40

FIGURE 5.1: AN ORGANISATION INCORPORATING SYSTEMS WITH LOWER
MANAGEMENT CAPABILITIES .................................................................................... 45

FIGURE 5.2: THE MANCONTRACTUAL PORT TYPE EXAMPLE .......................... 48

FIGURE 5.3: THE MANOPERATIONAL PORT TYPE EXAMPLE USING WS-
MANAGEMENT ............................................................................................................. 49

FIGURE 5.4: THE THREE MANAGEMENT SCENARIOS FOR ROLE PLAYERS .... 51

FIGURE 5.5: MANAGEMENT INTERFACES IN THE CONVOY SCENARIO ........ 51

FIGURE 6.1: DIRECT SUPERVISION IN ROAD SMC’S ......................................... 54

FIGURE 6.2: AN ABSTRACT POLICY CONTRACT IN A ROAD SMC ............... 56

FIGURE 6.3: MUTUAL ADJUSTMENT WITH CONVOYS AND THE RTA ............ 57
FIGURE 7.1: DEFINITION-TO-INSTANTIATION PROCESS IN ROADFACTORY......59

FIGURE 7.2: ROUTING AND RULE LAYERS IN THE SMC IMPLEMENTATION......60

FIGURE 7.3: MANAGEMENT MESSAGE ROUTING ..........................................................61

FIGURE 7.4: UML CLASS DIAGRAM OF PLAYER IMPLEMENTATIONS..................62

FIGURE 7.5: MANAGEMENT COMMUNICATION DEMONSTRATION SMC’S AND PLAYERS........................................................................................................................63

FIGURE 7.6: ABSTRACT POLICY CONTRACT DEMONSTRATION SMC’S AND PLAYERS........................................................................................................................66
List of Tables

**TABLE 5.1**: MANAGEMENT CAPABILITIES TO LEVELS OF AUTONOMY .............. 44

**TABLE 5.2**: MANAGEMENT INTERFACE PORT TYPES AND OPERATIONS .......... 48
List of Publications

This work is based in part on the following peer reviewed publication for which I am the primary author.

1 Introduction

In response to the ever increasing complexity of software systems, designing systems that are self-managed or autonomic has become a growing research area (Ganek and Corbi, 2003, Kephart and Chess, 2003). Systems with self-management capabilities help to reduce the burden on human operators, and abstract away large amounts of complexity inside reusable autonomic elements or self-managed components. These components are then composed into larger systems with goals greater than that of each individual part. One approach to embedding systems with self-management capabilities is to create systems as self-managed service compositions (Casati and Shan, 2001, Erradi et al., 2007, Pautasso et al., 2005). In this way the solution is more compatible with the existing principals of Service Oriented Computing (SOA), in that systems are still composed from discrete web services (as per SOA), but with the added concept of the composition itself taking on some of the management burden as an internal responsibility. In order to increase the abstraction of complexity, self-managed systems or more general service compositions are often functionality decomposed into hierarchies, with the higher level elements able to direct commands to the lower layers. In this way, each element of the system is only aware of the levels of the hierarchy directly above and below them, and only takes commands in a top down fashion. When organised in this fashion the structure of a system and its control relationships become intrinsically linked. Yet as systems become more open and exist across domains, there is a need to allow control relationships to exist independently of structure and for more flexible structures to be possible as opposed to just ridged hierarchies.

In business organisations, we form complex relationships and our organisational structures are rarely strict, ridged hierarchies. This allows us flexibility in the way we cooperate and undertake tasks. Relationships such as command-control rarely exist, but less formal types such as supervisor-subordinate and peer-peer are common place. Humans often are members of many social and business organisations at any one time, and those may contain specialised sub organisations that may in ways contradict the parent hierarchy, an example of which is a project or matrix style organisation (Skyttner, 2001). In these types of structures a hierarchy forms the basis of the organisation, but smaller work groups may form which have different goals and managers, while still existing as a part of the larger structure. The overarching goal
of this thesis is to further the ability to structure self-managed service compositions using complex organisational patterns and management relationships commonly found in business organisations. This entails a level of flexibility not often seen in software systems as well as a separation of control from structure.

1.1 Research questions

As systems become more open and distributed, these ridged organisational structures require more flexibility and granularity. If service compositions exist across organisational boundaries, simple command-control relationships may no longer suffice, as there may be conflicting obligations originating from different sources. If these compositions are self-managed, they may benefit from the less ridged organisations that we as human beings construct. Various patterns exist for modelling relationships between individuals and larger groups such as those described by Mintzberg (1983) and Skyttner (2001). If frameworks can exist that allow the creation of service compositions that mimic these patterns, then the systems themselves can also benefit from such flexibility as we do in business organisations. One such framework designed with this in mind is ROAD (Colman, 2006, Colman, 2007, Colman and Han, 2005a, Colman and Han, 2005b, Colman and Han, 2007, Colman and Han, 2005c). ROAD is a meta-model for designing self-managed service compositions. Further steps are required for the ROAD framework to make it a viable solution to realising the above mentioned goals. This thesis aims to address a subset of these problems.

The research questions to be addressed in this thesis are:

1. How to allow self-managed systems to exchange management communication that can facilitate the establishment of complex management relationships?

2. How to extend the ROAD framework to allow self-managed service compositions to be structured according to commonly found organisational management patterns?

1.2 The approach taken

Our approach to addressing the questions identified above is in the research context of the ROAD framework applied in a web services domain. The approach taken consists of the following four steps:

1. Firstly analysis has been conducted on several commonly found management structures and patterns found in organisational theory. Selections of these patterns are
identified as having relevance to software systems, in particular the ROAD framework.

2. The selected management patterns are expressed using the ROAD meta-model and areas of ROAD requiring additional work to better allow this are identified.

3. A research prototype is developed using the existing implementation of the ROAD framework which incorporates the additions indentified in step 2 and illustrates the management patterns using an example scenario.

4. The approach is evaluated to determine if and how well the management patterns have been incorporated and future work is identified.

1.3 Limitations and delimitations

This work in this thesis is the equivalent of one semester of full time work at Swinburne University of Technology. Due to this time constraint the work has the following limitations:

- Conceptually ROAD allows for potentially many kinds of entities to take part in a composition including agents. In this work we create a general solution but discuss how it may be applied to the web services domain.

- Only a small subset of organisational management patterns identified as being the most useful to the ROAD framework have been considered.

- Multi agent systems (which in areas draw some parallels to the work described here) are not discussed and are considered out of scope.

- The prototype implementation only serves to demonstrate the concepts described and is subject to improvement.

1.4 Research contribution

The contribution of the work described in this thesis is a furthering of the ability to structure software systems or specifically self-managed service compositions into flexible organisations. A subset of organisational management patterns are identified as being relevant to software systems. The management communication required to allow systems to enter into relationships that mimic the identified patterns is analysed and existing approaches
are evaluated on their ability to allow this management communication. The problems with these existing approaches are identified and the following steps undertaken to solve them:

- The development of a management interface which is to be used by ROAD to establish relationships with other services or service compositions (ROAD or otherwise) with varying levels of self-management capability.

- The development of an additional form of policy based ROAD contract which is designed to allow more flexible management structures from organisational theory to be modelled by ROAD.

- The development of a prototype implementation to demonstrate the feasibility of the above approaches.

As a side effect the work has also contributed to the development of the ROAD framework in general; including development of structural models that allow ROAD compositions to be defined declaratively and software to instantiate definitions into working software modules.

1.5 Thesis outline

This thesis consists of 8 chapters. Chapter 2 provides the background for the work conducted. Firstly it explores organisational theory / management patterns in human organisations and discusses which patterns are most applicable to software systems. Existing distributed management standards are then explored followed by policy based management approaches. Finally existing architectural frameworks supporting management relationships are explored and evaluated.

Chapter 3 describes a motivating scenario involving vehicles equipped with telematics systems, wishing to enter into collaborative route planning as well as forming convoys. This scenario is used to illustrate concepts throughout the remainder of the thesis.

Chapter 4 discusses an existing framework called ROAD, which can be used to design and instantiate self-managed service compositions. ROAD is used as a base framework for which this work is built upon.

Chapter 5 explores management communication that aims to allow the creation of complex management relationships for systems with differing levels of autonomy. Levels of management capability a system may possess are explored as well as the messaging
Chapter 1 Introduction

requirements of those levels. A management interface is defined for a web services environment which caters for the requirements of the various levels of capability. Finally this new management communication model is incorporated into the existing ROAD meta-model.

Chapter 6 describes a new form of ROAD contract called an abstract policy contract. This new contract aims to further the ability of ROAD to model management patterns explored in Chapter 2.

Chapter 7 describes a prototype implementation that aims to demonstrate the concepts described in the previous chapters. The prototype is built upon the existing version of the ROAD framework, which is briefly introduced.

Chapter 8 concludes the thesis by providing a summary of the work undertaken as well as highlighting its contribution to the area of self-managed systems and more specifically the ROAD framework. Limitations and future work are also noted.

Appendices A, B and C document the outputs of the prototype implementation, as well as example Drools rules and ROAD SMC XML descriptions.

1.6 Summary

In this chapter we have given an overview of the context of the work in this thesis, the research questions and limitations. We have also described the approach taken in addressing the research questions, as well as the overall structure of the thesis. In the next chapter we will discuss some of the background work and related material in organisational management patterns, as well as existing distributed management approaches from industry and academia.
2 Background

This chapter provides the general research background for this thesis. Section 2.1 examines management patterns in business organisations and identifies the types of management relationships that we require in order to further the creation of rich and flexible organisational structures in software systems. In Section 2.2 we go on to discuss work describing levels of autonomy individual components in a software system may possess which effects the position in an organisational structure those components may enter into. In Section 2.3 we examine existing software management protocols in some detail to determine if they have the expressiveness required to allow the creation of complex management relationships. In Section 2.4 we examine policy based management approaches, which in some ways mimic an important management pattern identified in Section 2.1. Section 2.5 discusses architectural frameworks that allow organisation of software systems into some form of management hierarchy and their ability to allow the complex management relationships described in Section 2.1. Finally we summarise in Section 2.6.

2.1 Management and organisational structures

In business organisations management is the coordination of activities and allocation of resources (human or physical) between entities (workers, departments, organisations) in order to fulfil specific tasks or achieve certain goals. In an organisation the responsibility of conducting management generally lies with a manager, a person who has been given the authority to make decisions and control subordinates. With management authority comes more responsibility. Managers may be entrusted to resolve conflicts or problems escalated to them by their subordinates, or enforce decisions placed upon them by other managers at a higher level in the organisation. Managers may also be held accountable for the actions of their subordinates.

Management can be conducted in a number of styles / patterns. Managers may be concerned with controlling the actions of subordinates in great detail (micromanagement). They may also simply assign tasks and trust they will be completed independently. Mintzberg (1983) defines five broad patterns of organisational management:
• **Direct supervision** shown in Figure 2.1(b) achieves coordination of work by having one person (the manager) take responsibility for the work of others. In this case the manager issues instructions to subordinates and monitors their actions closely. Here the manager has a constant responsibility to be aware of their subordinate’s activities and to directly coordinate those activities. The manager therefore needs detailed knowledge on how the work is conducted so they can coordinate the subordinates effectively.

• **Mutual adjustment** achieves coordination of work by allowing the control of the tasks to be completed to lie with the workers themselves. Here the manager assigns a task to subordinates initially, but then allows them to coordinate the work among themselves. The manager is then free to conduct other activities and need not worry about the details of what their workers are doing. This is illustrated in Figure 2.1(a). Here the activities of subordinates are somewhat abstracted away from the manager. The manager does not necessarily need to understand the method of how to achieve the end goal; they only need the ability to pass those goals down to subordinates and possibly receive status reports or updates on progress. This is analogous to a project manager of a team of software engineers. The project manager knows what the end product needs to be, but does not necessarily understand how to create that product, that responsibility lies with the software engineers. The project manager is free to concentrate on management specific tasks like budgets or schedules.

• **Standardised work process** refers to situations where the work to be completed is specified or programmed. An example might be when a worker is employed on a factory assembly line. They may have little contact with management as the instructions may be pre-specified for their position in the line, e.g. assemble each widget by connecting part x to part z and pass it to the next worker in the assembly line. This is illustrated in Figure 2.1(c). Here much like in mutual adjustment, the manager does not require detailed knowledge of what the subordinates are doing. The difference is that the level of skill required of the workers has been substantially reduced. The worker does not know how to achieve the end goal but has a set of instructions available which they can follow.

• If the required output is pre specified, but the process to achieve that output is not, this is an example of **Standardised outputs** shown in Figure 2.1(c). A good example of a
standardised output is when a taxi driver takes a passenger to a location. They do not
know the route to take beforehand but they do know their job is to get the passenger
to whatever location they desire and collect the fare on completion of the trip. Here
the requirements on the worker are heavier than that of standardized work process.
The worker is told the end goal and needs the ability to decide how to best fulfil that
goal without instructions.

- It is possible that neither the work process nor the output is standardised, but
  standardisation may still be required. Training a worker before allowing them to
  undertake their job is an example of **Skills and knowledge standardisation**.

![Diagram](image)

**Figure 2.1**: The five patterns of management (Mintzberg, 1983)

When put into the context of an organisational structure these management patterns are more
applicable at different levels of the hierarchy. Mintzberg (1983) also defines a basic structure
for an organisation which he splits into five distinct parts; Operating core, Middle line,
Strategic apex, Technostructure and Support staff. Figure 2.2 illustrates the main three of
these along with the direct supervision and mutual adjustment patterns at the most
appropriate levels.
Below we briefly describe the three core parts (as illustrated above) which are most applicable to this work.

- **The Operating core** of an organisation encompasses the areas which are directly responsible for work relating to production and services. This could be seen as the factory floor, where workers secure materials and transform those materials into some output or product.

- **The Middle line** is a series of middle managers whose authority increases as they rise the levels of the hierarchy. This chain of managers runs from the operating core to senior management. It is likely this not simply scalar, but a complicated structure in its own right, with some managers being subordinate to two or more senior managers and some being peers.

- **The Strategic apex** sits at the very top of the organisation. This area comprises a group of managers who are responsible for the overall direction and goals of the organisation.

When we apply the five patterns of management to this structure, we see that they are suited best to different areas. For example, a CEO of an organisation does not traditionally micromanage the senior managers who are his direct subordinates; he makes high level
Chapter 2 Background

decisions and trusts his subordinates to carry those out (mutual adjustment). As we travel lower through the middle line, the level of independence a manager has will gradually decrease, until finally in the operating core, a foreman may be directly managing a worker (direct supervision), or there may be some standardised process a worker follows. Therefore we can see the greatest levels of autonomy are needed at the higher levels of the organisation. Goals and policies put in place trickle down the management structure until they are realised by workers at the lower levels, essentially meaning the structure represents a hierarchy of decomposition as well to some degree a hierarchy of control. The control however is flexible, and is not strictly limited to command-control relationships.

Herbert Simon proposes hierarchy as ‘the universal principal of the structure of complex things’ where static hierarchical structures provide the key to solving complex problems (Agre, 2003). Hierarchies provide a natural way of solving problems by splitting and decomposing a problem into smaller more manageable parts. At any level some piece of the problem is solved, and the rest delegated to a lower level of the hierarchy. This does indeed imply a relatively static structure, where each entity blindly accepts tasks from a higher level, solves their part, and hands it down to the next level (a command-control relationship). A real organisation however does not function quite in this way. While hierarchies inevitably exist that promote some form of top down authority, the management relationships that exist more resemble the five more complex patterns described by Mintzberg. If we are to organise software systems into rich organisational structures that can express relationships beyond that of the hierarchy of decomposition, we need to at least allow for some these more complex patterns. There is an abundance of literature addressing organisational theory. In this thesis we will make use of the management patterns defined by Mintzberg, who is widely recognised as an authority on the subject. In particular we focus on direct supervision and mutual adjustment patterns. This is due to these patterns demonstrate in a very simple way the core principals seen in modern business organisations, without delving into more complex issues such as representing standardised work processes in a way software systems can interpret. They are also applicable to other organisational structures such as those described by (Skyttner, 2001).

2.2 Towards management structure in software systems

As we discussed in Section 2.1 position in an organisational hierarchy may influence the level of autonomy required to fulfil a role in an organisation. Roles higher in the organisation
require greater autonomy, while lower roles have requirements more concerned with the ability to follow instructions or commands. In self-managed / autonomic systems, the concept of varying levels of autonomy has been investigated in some detail. IBM\(^1\) proposes 5 levels of autonomy that an autonomic / self-managed element may possess. Each of these levels reflects the relationship between the element and its controller. As the levels increase, the skill required by the human controller reduces until the finally the element can act on its own in accordance with high level business goals. However the management relationship here is hierarchical in a sense that each element regardless of its level of autonomy, must comply with the orders of a human controller (command-control). The levels contain no notion of organisation or more flexible relationships. These levels are:

1. **Basic.** A basic element is managed by a skilled human controller / operator who utilises management tools to do so.

2. **Managed.** In a managed element the systems management tools can analyse and present information in such a way that it reduces the administration overhead on the human controller.

3. **Predictive.** Predictive elements build on managed elements in such a way that the management software is capable of recognising patterns and recommending courses of action to a controller.

4. **Adaptive.** An adaptive element uses the tools available to a predictive element but is able to make some changes independently of the human controller.

5. **Autonomic.** The final level, where systems are dynamically managed by policies and high level business goals. The system interoperates these and acts accordingly.

The adaptive level has been further broken down into sub levels by Huebscher and McCann (2008) which define whether the system or components management capability is focused on a single aspect of the system or a more end to end solution. This however has no real impact on our discussion here.

Chapter 2 Background

Viable systems architecture (VSA) (Herring and Kaplan, 2000, Herring, 2002) defines levels of conceptual development in a software system, each being another evolution of the base system process towards becoming self-managed, there are:

1. **Basic Control**, where a simple feedback controller is added to the running process that allows a human to give commands and observe the results. The environment is considered as a source of disturbance to the process, which may also impact the environment in some way. More than one process may take commands from a single controller.

2. A **Regulator** is added that contains explicit representation of plans, time tables etc. This is linked between the controller and the running process in order to provide some level of coordination. Again there may be multiple regulators added for a single running process.

3. An **Auditor** is introduced along with the regulator that may accept audit requests from the controller and return reports.

4. The second major step in reducing human control of the system is the introduction of an **Adaptive Controller**, which resides above the previously added controller. Now the human may give rules instead of just commands which the adaptive controller will see enacted. The adaptive controller may also monitor the environment to determine when rules need to be executed.

5. The final addition to the abstract architecture is the inclusion of a **Supervisor**. The supervisor now only takes high level policies and goals from the actor and interacts with the adaptive controller. The system may now be considered self-managed.

These levels of evolution do not in any way model relationships between systems, regardless of the level of autonomy achieved. VSA is further discussed in Section 2.5.

Colman and Han (2007) define levels of autonomy that describe the relationship of a component with an organisation via an organisational role, analogous to a job description in a business organisation. The component in this case is referred to as a player (as it plays the role). These levels impact on the detail that must be provided to the player, e.g. a player with a low level of autonomy may require a process description in order to fulfil its goals while a
player with a high level of autonomy may simply require some high level goal or policy (similar IBM's level 5). The five levels defined here are:

1. **No autonomy.** The player follows a process description defined by the position in the organisation.

2. **Process autonomy.** The player can choose a process to meet the required system state.

3. **System state-autonomy.** The player can choose a system state and process necessary to fulfil a goal defined by the organisation.

4. **Intentional autonomy.** The player can choose whether or not it will fulfil the goal set forth by the organisation. A player with this level of autonomy is what we refer to as independent.

5. **Autonomy from constraints.** The player is prepared to violate any constraints or rules to achieve its goal, e.g. use system resources at the expense of others. This sort of player should be generally avoided.

None of these approaches address the problem of describing management relationships between elements of the system. Using the levels defined by Colman (2007) we can describe position descriptions that require varying levels of autonomy within an organisation as roles. Positions near the top likely require the levels such as intentional autonomy or autonomy from constraints, while lower positions require one of the first 3 levels. The next step is to find a way to describe the relationships between those roles in such a way that they can be structured as per the management patterns discussed in Section 2.1 to help form richer organisational structures. As the field of autonomic systems progresses, components with a greater level of autonomy will inevitably be created. The ability to express more complex management relationships between them will allow us to make better use of these capabilities.

### 2.3 Management in existing software systems

Management is a concept that has existed in the software world for some time. Typically operational management of resources (fine tuning, configuration, monitoring) is becoming more common place as computing environments become more distributed and service oriented. We define a **resource** as any software or hardware based asset within an
organisation. Resources in an organisation may be located in a remote area, creating the need for distributed management protocols to exist so the resources can be monitored, configured, and repaired from any location. If a resource is capable of allowing this remote management, we refer to it as a managed resource. The people who have control over a managed resource could range from administrators, developers or organisational managers. These people are not the end users of the resource, but have responsibility to ensure it provides the service or function for which it is intended. We will refer to anyone with this responsibility as a controller. The end user of a managed resource is referred to as a consumer. There exists the potential that a consumer of a resource may also have control of that resource, meaning the resource could be tweaked or finetuned during operation, potentially making the resource far more useful. This however is not usual pattern seen when distributed management protocols are put in place.

Typically management protocols allow for three main operations regardless of whether they are designed for software or hardware resources.

- **Resource queries** allow a controller to retrieve information from a managed resource. For example the current state of the resource could be queried or more specific information such as IP address or name.

- **Resource updates** allow a controller to change some value on the managed resource. This could be some form of state change or a simple configuration change. Generally any value that can be read via a resource query can be set via a resource update.

- **Topic subscriptions** allow controllers to subscribe to any number of topics that are offered by the managed resource. This allows asynchronous notifications to take place in notification of some value on the managed resource changing. The typical use for this operation is subscribing to a change in state that would indicate a fault has occurred.

The three operations are illustrated below in Figure 2.3.
Below we describe the most common management protocols and standards currently being deployed in computing environments. While somewhat relevant, we do not include agent-oriented approaches in this survey. Agent oriented systems consist of collections of individual agents which coordinate tasks among themselves based largely on their own deliberations. Efforts exist into organising agents into organisational hierarchies (Vazquez-Salceda et al., 2005, Wise et al., 2000) but they do not allow for the complex management structures we are aiming to create here. Some of the general concepts of management communication defined later in Chapter 5 do however draw parallels to the FIPA-ACL\textsuperscript{2} specifications.

2.3.1 Simple Network Management Protocol

Simple Network Management Protocol (SNMP) (Case et al., 1999) is an application layer protocol developed by the Internet Engineering Task Force (IETF) to be used over TCP/IP networks. It was designed with network devices in mind such as routers, switches, hosts and gateways. SNMP has become the industries defacto standard for network management.

SNMP’s general domain of use is to allow network administrators to manage network performance and find / solve network problems. An SNMP network consists of three components; managed devices (analogous to managed resources), SNMP agents, and network management systems (NMS’s). Managed devices are network enabled devices that contain an

\textsuperscript{2} The Foundation for Intelligent Physical Agents, \url{http://www.fipa.org/} (viewed May 2009).

---

**Figure 2.3:** The three standard management protocol communication patterns
SNMP agent. An SNMP agent is a software based component that has access to local knowledge of the devices management information that can be accessed / manipulated. An NMS is a client application that connects to remote SNMP agents to allow a human controller to collect and modify management information. One or more NMS’s will usually exist on any managed network. Figure 2.4 shows an example of an SNMP enabled managed network.

![Figure 2.4: An overview of a SNMP enabled network](image)

The Management Information Base (MIB) is a database that exists on SNMP managed devices and stores management information as key / value pairs. The values are organised into a hierarchy and are identified by object identifier’s (OID’s). The top level MIB OID’s belong to various standards organisations, while lower OID’s are allocated by associated organisations. Vendors can define custom branches on OID’s in the hierarchy that include non industry standard information for their own specific products.

SNMP devices can be monitored and controlled using the same three basic communication patterns we have previously identified; queries, updates and subscriptions. SNMP provides a fourth pattern which is a traversal of the MIB hierarchy from a specified starting point in order to retrieve many related values at once.
Chapter 2 Background

Various tools exist that implement SNMP based NMS systems such as HP OpenView\(^3\) and OpenNMS\(^4\). SNMP4J\(^5\) is an API written in Java which provides a way for Java applications to make calls to SNMP enabled devices. Most modern operating systems provide built in SNMP agents to allow PC and server software and hardware to be managed by an NMS. Some other purely software based products also have SNMP agents such as the Java virtual machine.

SNMP is essentially a standard and accompanying protocol and it is up to manufactures to implement SNMP agents and MIB’s for their devices. The protocol is very closely tied to device implementation which has prevented it from moving forward due to backward compatibility issues (Papazoglou, 2008).

2.3.2 Common Information Model / Web-Based Enterprise Management

Common Information Model (CIM) and Web Based Enterprise Management (WBEM), developed by the Distributed Management Task Force (DMTF), are standards used to describe and communicate elements (analogous to managed resources) across an enterprise, including hardware and software systems (Hobbs, 2004, Papazoglou, 2008). CIM is structured in such a way that the managed environment can be seen as a collection of interrelated systems, each composed of various elements. CIM supplies classes with properties and associations that provide an object oriented (OO) view of the managed environment, the OO view providing a commonly understood modelling paradigm. CIM is structured into three layers:

- **The core model.** An information model that captures notions that are applicable to all management areas.
- **The common model.** An information model that captures notions that are common to particular management areas but are independent of a particular technology implementation. The common areas are systems, applications and devices. This schema provides a set of base classes for extension into more specific technology

---


areas. The core and common models are commonly just referred to as the CIM schema.

- **Extension schemas.** Technology specific extensions to the common model schema, e.g. for Microsoft Windows or UNIX systems.

CIM and WBEM were originally separate standards but CIM now exists as a sub standard to WBEM. WBEM consists of three components; CIM which provides the format for describing management data, xmlCIM, which provides XML elements for describing CIM classes and instances, and finally the CIM over HTTP specification which provides a transport mechanism.

Using CIM over HTTP, xmlCIM can be used as a payload in a HTTP message. CIM operations over HTTP can be classified as; basic read, basic write, instance manipulation, schema manipulation, association traversal, query execution and quantifier declaration. These differ from SNMP operations as the controller now has the ability to not only query management values, but manipulate the class model at runtime as well.

### 2.3.3 Web Services Distributed Management

Web Services Distributed Management (WSDM) (Bullard et al., 2006) is an OASIS specification for management of distributed managed resources. WSDM takes advantage of existing web service technology in order to create management interfaces for resources. In order to make use of WSDM, a resource must implement the management mechanisms and a web service interface to expose them. WSDM is divided into two sub standards, Management using Web Services (MUWS) (Bullard and Vambenepe, 2006), and Management of Web Services (MOWS) (Wilson and Sedukhin, 2006). MUWS will be described here as it provides the key standards needed to manage resources in general, not just specifically web services. Any overlapping information between the two standards is included in MUWS.

MUWS specifies how the manageability of a resource is to be made available via web services. The web service for a managed resource which acts as a management interface is called a manageability endpoint. Like other standards MUWS does not define an implementation strategy for manipulating and retrieving management information from a managed resource, only a specification for the management interface and a message format. A sensible implementation would be to follow the SNMP approach and separate the managed
resource and the management interface via some agent with the capability to translate management messages and manipulate / query the resource.

MUWS makes use of a suite of standard web service technologies in a layered model. As with web services in general the underpinning technology of WSDM is XML, as well as SOAP and WSDL. At the next level MUWS defines a set of standards dictating how to represent a managed resource as an XML document. These standards dictate that every resource be exposed by a resource properties document. A resource property exposes some form of management information that is a part of the state model for a managed resource.

Once again MUWS follows the three standard communication patterns we have previously defined. To delve in a little further, MUWS allows a controller to get / set single resource properties, get / set multiple resource properties, and query relationship information (relationships with other resources in which the resource participates).

2.3.4 Web Services for Management

Web Services for Management (WS-Management) (Arora et al., 2004) is a standard developed by the DMTF and is a direct competitor to the WSDM standards. Both standards have their similarities, such as use of standard web service technologies (XML, SOAP and WSDL). WS-Management however seeks to achieve similar goals with a smaller footprint. The standard defines a minimal set of specifications and usage requirements, and leaves it to the developer to further extend these as necessary. WS-Management again allows for our three communication patterns (queries, set values and subscribe), but also states two others; enumerating the contents of containers and collections and custom management operations. The ability to define custom operations with any desired strongly typed input and output parameters sets WS-Management apart from WSDM, which is much stricter and focuses on the querying and updating of individual values.

IBM has announced they are planning to develop a common specification by converging the WSDM standards and WS-Management. It is debatable as to which of the standards is currently more popular and both have various implementations\(^6\)\(^7\).

---


\(^7\) Wiseman, [http://wiseman.dev.java.net/](http://wiseman.dev.java.net/) (viewed November 2008).
2.3.5 Java Management Extensions

Java Management Extensions or JMX (Perry, 2002) has existed as a part of the Java platform since J2SE version 5.0. JMX provides a very simple way to manage resources such as applications and devices. Using JMX, a given managed resource is instrumented by one or more Java objects known as Managed Beans or MBeans. The MBeans are registered in managed server known as the MBean server, which acts as a management agent and runs on the Java virtual machine. A managed resource could potentially be a single Java class, many classes, or some device or application which has been abstracted by an MBean.

JMX defines standard connectors that allow access to JMX management agents remotely. JMX connectors using different protocols provide a standard interface, so a management application can manage a resource transparently. This potentially means an MBean could be interacted with via the SNMP or WS-* standards.

The JMX architecture is separated into three distinct layers (illustrated in Figure 2.5):

- **Instrumentation.** Managed resources such as applications, devices and services, are instrumented using MBeans. MBeans act as the management interface, which exposes attributes and operations via a JMX agent.

- **Agent.** The main component of a JMX agent is the MBean server. This is a core managed object server in which MBeans are registered. A JMX agent also includes a set of services for handling MBeans. JMX agents directly control managed resources and make them available to remote management controllers.

- **Remote Management.** Protocol adaptors and standard connectors make JMX agents accessible from remote management applications outside the agents JVM.

![Figure 2.5: JMX architecture overview](image)
2.3.6 Web Service Offerings Language

The Web Service Offerings Language (WSOL) (Tosic et al., 2005, Tosic et al., 2002) is an extension to WSDL that allows formal specification of classes of service called service offerings. A class of service is a variation on the normal functional and quality of service aspects of a web service. The concept of class of service allows a web service to differentiate itself based on the differing requirements of consumers, or based upon the price a consumer may be willing to pay. This may be achieved by placing constraints on different aspects of a web service, such as response time, access rights, the type of devices it may execute on, etc. A benefit of this is the ability to define different classes of consumers in order to balance usage and keep a web service operating at 100% utilisation. If a consumer is willing to pay more they receive a premium service, while others may wish to minimise their price / performance ratio.

WSOL allows specification of port-level service offerings, which specify constraints only upon the constructs in the specified port. A component level service offering can be applied to a web service with multiple ports and specifies the allowable combination of port level service offerings. Relationships between offerings may also be defined for transition if one offering cannot be maintained.

2.3.7 Evaluation

The first five management standards described above (SNMP to JMX) all follow the same basic pattern of management, which is a command control relationship. Another way to commonly refer to them is as operational management standards. A given controller (traditionally human) may only communicate simple commands to a resource and the resource is expected to comply in full as it possesses no real autonomy of its own. These standards allow a human controller to perform basic operations on a managed resource that are designed to facilitate monitoring and fault recovery, not communication between elements with varying levels of autonomy. The standards do not take into account the fact that in a self-managed system, different elements may have differing goals. An element with a high level of autonomy may have the ability to reject any commands it receives from a controller if it does not deem them in line with its own goals.

WSOL provides a mechanism for providing different classes of service called service offerings but provides no mechanism to negotiate offerings or dynamically create them based on management relationships. If we are to represent complex management relationships, we
need a standard that allows; high level goal communication, negotiation, and agreement formulation / maintenance. This is not to say however that the existing management standards described above may have no use in a system composed of autonomic elements. Elements with low levels of autonomy may be in a position where their capability only allows operational management, such as in the operating core described in Section 2.1. This would be sufficient as systems in this lower level merely need to accept commands and execute tasks. A system with a higher level of autonomy however will need to enter into some form agreement that stipulates it is willing to accept commands. Standards like WSOL also may help support decision making in dynamic / adaptive service compositions, but lacks the infrastructure to use service offerings as a basis for entering into complex management relationships. An existing standard therefore should be extended, so that it encompasses operational management capability as well a more complex communication to allow the creation of complex management relationships.

## 2.4 Policy based management

Methods exist for managing software components based on policies as opposed to the more direct methods of management discussed in Section 2.3. Policies define how an element in a certain domain may act e.g. their requirements, what resources they have access to and what behaviour must they conform to. This makes policy based management approaches particularly applicable to this work as they mimic management patterns like mutual adjustment. This section will provide a brief introduction to some of these methods, namely WS-Policy, Ponder, and Rei. Other policy based management approaches such as ITEF (Moore et al., 2001) and KAoS (Uszok et al., 2004) are summarised by Phan, Han et al (2008). These approaches are not described here as conceptually they offer no benefits over the others when evaluated on how well they may be able to be used to represent the mutual adjustment pattern.

### 2.4.1 WS-Policy

WS-Policy (Box et al., 2004) is a policy specification represented in XML that is developed for use with web services. It provides the grammar necessary to represent the characteristics and requirements required of entities in a web service based system. WS-Policy defines a policy to be set of policy assertions. These assertions are grouped into alternatives by operators (based on Boolean logic) that define the necessary assertions and / or combination or assertions that must be adhered to. Policies that can be represented using WS-Policy range
Chapter 2 Background

From traditional requirements that manifest on the wire on to others that have no wire manifestation but may still be considered critical such as privacy. WS-Policy is simply a specification for representing policies, how the policies described using WS-Policy are actually implemented and interpreted is something that is left to the developer. WS-PolicyAttachment (Bajaj et al., 2006) is a related specification which defines how to associate a policy with a accompanying web service endpoint.

2.4.2 Ponder

Ponder (Damianou et al., 2001) is a declarative object oriented language for specifying management and security policies, for distributed systems coupled with a deployment environment. Ponder allows policies to be enforced on subjects, which are users and administrators of the system. A subject’s policy is in relation to target objects, which are resources in the system such as servers, databases, etc. Policies are applied to one or more domains, which are groupings of subjects and objects into related areas. This allows a policy to be applied to a domain regardless of the subjects or objects in that domain. Once an object or subject is placed into the domain it automatically becomes subject to any policies already applied. Domains are similar to directories and are implemented using LDAP (Lightweight Directory Access Protocol).

Ponder supports access control policies which govern the activity of a subject after being authenticated by the system. The types of access control policies supported are:

- **Authorisation** policies, which define the activities a subject may perform on objects.(Box et al., 2004)

- **Filtering** policies that transform input and output in an action. For example an employee in an organisation may have access to a directory of other employees which contains detailed information. A user of the directory from outside the organisation may only have access to basic contact information. This is a good example of applying different policies based on domain.

- **Delegation**, which is the temporary transfer of access rights allowing authorised users to gain privileges temporarily.

- **Refrain** which defines the actions a subject must not perform on an object.
Chapter 2 Background

Ponder supports the ability to group policies based in common patterns to promote reusability. It is also possible to define roles, generally pertaining to a position in an organisation. Policies can then be applied to roles regardless of the actual subject assigned to those roles. Policy based relationships can be defined which govern obligations between two roles. This can potentially be used to structure subjects into some form of management structure e.g. a policy could be defined to specify that the subject assigned to role A must send reports to the subject assigned to role B at regular intervals.

2.4.3 Rei

Rei (Kagal, 2002) (which means ‘universal’ in Japanese) is a policy specification language based on deontic constructs. The language aims to be domain independent but allow the incorporation of domain dependent information easily. Rei has been developed for use with the Me-centric project, which is an approach to developing context aware systems. Me-centric divides the system into context based domains which may potentially overlap, much like the concept of a domain in Ponder. Rei allows for policies based on rights, obligations, prohibitions and dispensations. Associated with the policy language is a policy engine which interprets and reasons about policies and makes decisions.

At the core of Rei are policy objects. These are the constructs used to describe policy conditions and actions. These are represented in a simple first order language, e.g.

@\(\text{action, condition}\)

The @ is substituted for rights and prohibitions or obligations. These are then associated with the subjects of the system in a similar way, e.g.

\(\text{has(subject, policy object)}\)

An example of this may be \(\text{has(AgentA, right(printAction, hasPrintCredit(Agent)))}\) which means AgentA has the right to print but only when they have print credit.

Rei aims to make the language for defining policies much simpler than others like Ponder, which use complicated syntax that differs based on the type of policy being defined. Rei also allows for groups of policies to be defined as well as the concept of roles much like Ponder.
2.4.4 Evaluation

Policy based management approaches allow us to govern the access to resources in policy domains, separating the policy constraints away from the resources themselves. This effectively creates a policy layer which can be overlayed on top of existing systems, facilitating separation of concerns. Policy languages like Ponder attempt to address the problem of structuring resources into management relationships based on policies. This is accomplished by constraining communication / message types allowed between two components and nominating some basic obligations one may have to the other (e.g. send a report once a month). However defining the obligations between two resources is not sufficient to allow us to structure software systems into rich organisational structures. Mechanisms will need to exist to define the managers who have responsibility over relationships between other elements in the system and what level of responsibility they have to each other and the manager. This will allow us to represent complex management patterns such as mutual adjustment. If parties in a relationship are self-managed, they may have conflicting goals, which may result in the need for problems to be escalated to their assigned managers. The communication is therefore very dynamic. The message flow between any two resources cannot simply be predefined using the methods provided by Policy based approaches. Policy based approaches also contain no notion of an organisation. Behaviour of elements may differ based on the policy domain they reside in, but this is not expressive enough to create the organisational structures discussed in Section 2.1.

2.5 Architectural frameworks supporting software management structures

There has been an abundance of previous work which attempts to address structuring software systems into management hierarchies at some level. These provide an infrastructure around structuring self-managed systems into command control style relationships. This section briefly discusses a selection of these that most relate to this work, namely LIRA, VSA,

2.5.1 Lightweight Architecture for Reconfiguring Applications

LIRA, a lightweight infrastructure for reconfiguring applications (Castaldi et al., 2003) extends the concepts of network management standards, in particular SNMP, to component based distributed systems. LIRA defines two types of components or agents:
Chapter 2 Background

- A **reconfiguration agent** is responsible for managing a component in response to operations it exposes. It defines a MIB that is interacted with via SNMP.

- A **host agent** is associated with a computer in a network and is responsible for installing and activating components on that computer. It also responds to operations on variables in its MIB.

Base reconfiguration agents can become **managers**. These manage a subassembly of reconfiguration agents. In this way recursive hierarchies can be constructed as sets of manager to base reconfiguration agent relationships as illustrated in Figure 2.6.

A reconfiguration agent exposes at least five management operations; start, stop, suspend, resume, and shutdown. Two variables are also exposed; status, and notifyTo. Status contains

![Figure 2.6: An example LIRA agent hierarchy (Castaldi et al., 2003)](image)

the state of the component while notifyTo allows communication with sub reconfiguration agents in order to distribute commands. Additional operations can be added for domain specific activities. Host agents expose similar operations but also allow for the installation and un-installation of components and agents.

### 2.5.2 Viable Systems Architecture

Viable systems architecture (VSA) (Herring and Kaplan, 2000, Herring, 2002), is based on the Viable Systems Model (Beer, 1984, Beer, 1985) which seeks to capture the invariants required by successful software systems (called viable systems). The goal of VSA is to allow
growth, so that the software can be adapted (design time by humans) towards becoming an adaptive system (self-managing).

The architecture is illustrated in Figure 2.7, and incorporates the levels discussed in Section 2.2 which describe the evolution of a basic software process into a self-managed software system.

![Diagram of VSA architecture](image)

**Figure 2.7**: A system designed using the VSA architecture (Herring, 2002)

A system designed using VSA is referred to as a viable component. Viable components may be constructed recursively, where the running process being managed is another viable component and the outputs of the auditor and regulator become inputs to the sub components supervisor. Multiple viable components may also be positioned under single regulator and auditor. As well as this hierarchy viable components may be positioned in a peer-peer style relationship so that they may coordinate, although there is no explicit representation of the allowable interactions or obligations between two components.
2.5.3 Evaluation

The architectures discussed allow the structuring of components into some level of management relationships. LIRA is designed to be explicitly hierarchical and only allows command control relationships. Each reconfiguration agent sends commands to those in the level below. No agents may be considered peer to peer and each must conduct its actions based on its position in the hierarchy, with no room for negation or agreement formulation. VSA extends this in that each viable component in a hierarchy has a supervisor component which depending on its level of intelligence (which is left open in VSA) may possibly reject a goal or policy from a higher level. Peer-peer style communication is also allowed for but only in a cursory manner. As it is a high level architecture it lacks the explicit representation of agreements that may stipulate exactly the form of management relationship that exists between components. It also lacks an explicitly defined way in which components may come to form these relationships. These attributes are required in order to structure the components into complex management relationships. ROAD is a framework that does explicitly represent the agreements between two parties, but does not represent a way in which parties come to form these agreements. ROAD is discussed in detail in Chapter 4 and forms the basis for the remainder of the work in this thesis.

2.6 Summary

In this chapter we have presented the relevant background material to the work described in this thesis. In Section 2.1 we gave an overview of commonly known organisational management patterns and an overview of the hierarchical nature of business organisations. The patterns identified serve as a basis for those that need to be expressed in software systems organised using complex management relationships forming a rich organisation. Section 2.2 described previous approaches to describing the levels of autonomy an element in a self-managed system may possess. These levels describe the relationship of an element with its human controller, and the relationship an element may have with an organisational role. They do not address the need to describe the relationships between the elements themselves. An overview of the most commonly used software management standards / proposed standards is provided in Section 2.3 and policy based management standards in Section 2.4. These standards are designed to allow human controllers to maintain distributed components in software and hardware systems and in the case of policy based techniques, to constrain their behaviour based on the policy domain they are placed in. They however provide no
means to describe complex management relationships between elements of the system, nor do they take into account the differing levels of autonomy an element may possess. Other related work on architectural frameworks that allow organisation of software systems into some form of management hierarchy is discussed in Section 2.5 In the next chapter, we present a brief motivating example scenario that will be used throughout the remainder of the thesis.
Chapter 3 A Motivating Example

3 A Motivating Example

This chapter presents a motivating example scenario that will be used to illustrate important concepts throughout the remainder of this thesis. Vehicle interaction using wireless networks has been identified as a research challenge and future growth area in the automotive industry (Broy, 2006). Vehicle interaction provides a good example of the types of management relationships that may exist between independent systems in dynamic open environments. The scenario will be concerned with multiple cars travelling together from Melbourne to Sydney and is adapted from existing work. These cars are each equipped with a conceptual telematics system that allows cooperative route planning as well as other collaborative modes of operation. We assume the technology being used to facilitate communication between systems is web services. In this chapter we firstly describe the cars telematics systems ability to form a convoy, allowing one car to take on the role of a leader and the remaining cars as followers. We then describe a Road Traffic Authority (RTA) system which can impose behavioural constraints on the travelling cars if they agree to allow it.

3.1 Cooperative route planning and convoys

Jane, Bob and Tom are drivers of three cars travelling together from Melbourne to Sydney. Each car is fitted with a telematics system allowing them to communicate information. The systems allow for the following modes of operation:

- **Cooperative route planning mode.** This allows the drivers of the cars to modify a route to their destination via a map on a screen on their dash boards. This mode acts much like a collaborative white board application, in that any changes to the route on one cars screen will also cause the same information to be changed on any participating cars screens.

- **Convoy mode.** In this mode, one car searches for the best route and propagates this to other cars. The car searching for the route takes the role of the *leader*, while any others receiving route information are *followers*. Convoy mode has a number of

---

8 Thanks to Dr Minh Tran of Swinburne University for the use of his scenario.
constraints. First, the following cars must notify the leader car of their distance from
the leader car every 10 seconds (which would be automated). When this distance
becomes greater than 300 metres, the cars are notified that they must rectify this.
Failure to do so may result in the follower car no longer being considered a part of the
convoy. Next, the leader car must update any route changes to the followers, no
matter how small. Finally, in the event of detioriating weather conditions, e.g. a storm,
the distance between the leader and follower cars must be reduced to 100 meters.

• Fault / problem notification mode. This mode allows cars to notify each other if any
problems occur that may force them to stop or divert course, for example someone
becoming car sick may force a car to pull over for a period of time. It also allows
automatic notification of mechanical failures to other cars if they can be detected.

Convoy or cooperative mode may be used alongside fault / notification mode. A combination
of convoy and cooperative mode may also be employed, allowing follower cars to make route
change requests to the leader. These may be rejected, or accepted and propagated to other
follower cars. The constraints and obligations defined in these modes of operation are not
pre-determined constraints imposed by a software engineer; they must be defined at runtime
by some form of agreement negotiated between participating cars depending on their systems
level of autonomy, the driver’s wishes, as well the general capabilities of the cars. They are
also highly dynamic and can be re negotiated at any time by either party if some condition
can no longer be met. We can see these relationships are not command-control, they are
much more flexible and each party is assumed a level of autonomy, whether it is in the
drivers of the cars or an intelligent software system. Existing approaches identified in Chapter
2 do not allow these sorts of relationships to be established, nor do they allow any way to
represent and maintain them.

3.2 Road traffic authority

The Road Traffic Authority (RTA) has a system in place to monitor various roads for
accidents and dangerous weather conditions. This system can also interact with cars that
possess a compatible telematics system if they agree to take part in the RTA system. The
RTA system has the following capabilities:
Chapter 3 A Motivating Example

- **Accident notification.** The RTA monitors the road for reports on accidents, whether they are due to a car crash or some other disruption (fallen trees, floods, fires, etc). Cars in the area can then be notified so that they may adjust their routes accordingly.

- **Weather condition monitoring.** Changing weather conditions may affect how fast a vehicle may safely drive on the road. The RTA system can notify cars of recommended speed changes, e.g. when clear a roads speed limit may be 100 kilometres per hour, when raining it may be recommended to reduce speed to 80 kilometres per hour.

- **Convoy constraints.** Depending on the road cars are travelling on, there may be constraints on the types of convoys being permitted. For example on a busy freeway in peak hour a funeral procession consisting of multiple cars, travelling slowly and in close proximity will likely cause a traffic jam. The RTA may stipulate that convoys on certain roads must be spaced a certain distance.

In their trip between Melbourne and Sydney, the drivers have decided to create a route together in cooperative mode. After this they will form a convoy, in which it is negotiated that Jane is the leader and Bob and Tom are followers. All drivers have registered themselves to the RTA system and will be notified of weather, accidents, and convoy constraints. In this way they have agreed to abide by the restrictions imposed by the RTA despite the fact they are themselves autonomous entities. With this fact in mind, they may choose to opt out of receiving RTA constraints at any time.

We can see in our example, cars may communicate with one another independent of the RTA system. They may also communicate with the RTA regardless of each other. The RTA however has the ability to impose constraints on the relationships among cars, e.g. convoy distance must be greater than 50 metres etc. Here we see some examples of the management patterns discussed in Chapter 2. The RTA possesses a higher authority than that of the cars and has responsibility over the use of the roads. In this way it can be viewed as a supervisor or manager over the cars registered to its system. It imposes goals or desired states on the cars registered in its system, e.g. distance kept between them, accident and whether recommendations etc. The RTA does not however actively monitor the cars and therefore the relationship is much like mutual adjustment. If the RTA were to communicate directly with each of the cars and monitor their compliance in some way it would more closely resemble a
Chapter 3 A Motivating Example

direct supervision style of management. We also see examples of peer-peer (cooperative route planning), and supervisor subordinate between cars (convoy mode).

An important point is that these cars are completely independent of one another. No car can be forced to take part in a convoy, or obey instructions from the RTA. Agreements need to be established between parties that stipulate the allowable interactions and obligations on those interactions (e.g. send distance updates every 10 seconds, obey constraints imposed by RTA, etc).

3.3 Summary

In this chapter we have presented a motivating example involving cars with telematics systems travelling from Melbourne to Sydney. These cars wish to interact with one another to form a convoy and exchange route information as described in Section 3.1. The cars are considered autonomous entities and the relationships between them highly flexible and dynamic. An RTA monitoring system also exists which can interact with the cars and impose constraints on the relationships between them if they choose to allow it as described in Section 3.2. We have also pointed examples of the management standards discussed earlier and noted that the existing approaches described in Chapter 2 do not allow creating systems of this kind. The example will be used throughout the remainder of the thesis to illustrate important points. In the next chapter we introduce ROAD, a framework for designing self-managed service compositions that unlike previously mentioned approaches contains constructs that allow creating dynamic and flexible management relationships.
4 An Introduction to the ROAD Framework

In this chapter we give a brief introduction to the ROAD framework and its accompanying implementation, which serves as a basis for the work in this thesis. ROAD is made up of a number of essential components, most notably roles and contracts, each of which are discussed in Section 4.1, followed by a discussion of the current state of the ROAD implementation in Section 4.2. We go on to demonstrate the motivating example from Chapter 3 in the context of ROAD composites in Section 4.3, and summarize in Section 4.4.

4.1 The ROAD meta-model

ROAD\(^9\) (Colman, 2006, Colman, 2007, Colman and Han, 2005a, Colman and Han, 2005b, Colman and Han, 2007, Colman and Han, 2005c) is a meta-model for defining self-managed service compositions as flexible organisational structures called self-managed composites or SMC’s. ROAD SMC’s are adaptive as the relationships between services can be regulated at runtime. The underlying principals of ROAD are; the separation of control from structure, the distribution of control over recursive hierarchies, and the separation of a role definition from its actual implementation. An SMC consists of a number of essential meta-model components; roles, role players, contracts, and the organiser. These have been formalised into sets of XML schemas that can be used to define SMC’s declaratively (discussed in the next section). The essential components are explained below:

- **Roles** (also called functional roles) are first-class runtime entities that define a position description within the organisational structure of an SMC. Roles expose allowable operations that can be invoked as well as the required operations of any potential role players, in this way a role resembles a WSBPEL partner link\(^10\) (required and provided interfaces). Roles also contain message queues so that an SMC may

---


continue to operate even if a role in left unfilled at any time. In the ROAD meta-
model roles may also describe non functional requirements but this is not considered
here.

- **Role Players** of functional roles are entities able to carry out the functions defined in
  a role. This is analogous to a job description in a business organisation, e.g. a
  receptionist. The job description of the receptionist role exists regardless of whether
  an actual person is employed to take on that position. The aim of ROAD is to allow
  any kind of entity to become a role player e.g. web services, agents, graphical
  applications for human interactions, etc. Currently development is focused on a
  version of ROAD for use in a web services context. ROAD SMC’s may also play
  roles in other SMC’s, allowing for the creation of recursive hierarchies.

- **Contracts** are used to link roles together in an SMC. A contract defines the allowable
  interactions and obligations on those interactions between two parties. As contracts
  define the allowable interactions, the allowable operations on a role are dependent on
  the contracts linked to it. Contracts are complex entities in themselves that
  encapsulate constructs like terms, obligations, and clauses which are explored in
detail by Singh (2008) and Colman (2007). Performative contracts also exist that
  make use Control-Communication Act (primitives) to constrain functional
  information flow between parties which in a way resemble the REI policy constructs
described in Section 2.4.

- **The Organiser** is an entity that is responsible for the self-management aspect of an
  SMC and interacts with the SMC via an organiser role. Like a regular player, the
  organiser can potentially be anything that has the ability to fulfil the operations set out
  by the organiser role. In this case these operations allow the organiser to add / remove
  contracts and roles, alter contracts, and bind / unbind role players. ROAD as a project
  focuses on the architecture required to create adaptive organisational structures. The
  intelligent entity that fulfils the organiser role is not a focus of the project and is
  currently considered a black box.

An example of a ROAD SMC is illustrated in Figure 4.1. The SMC models the relationships
between the various parties involved in the stock exchange. Clients may contact brokers in
order to buy shares in various companies. Clients may never interact directly with companies;
brokers take on this role and act as a middle man between the two. The contracts binding clients and brokers may contain terms that allow the placing of orders for buying and selling shares, with obligations defining time frames for settlement if a transaction occurs (3 days in the Australian stock exchange). Brokers and companies have their own set of respective terms and obligations that bind them, such as the actual buying shares if they fall within the client’s price range, invoices and required documentation exchange, etc. The players of these roles may then be anyone capable of fulfilling the required interactions and obligations. The organiser is also illustrated for completeness.

Using contracts and roles in ROAD gains a level of flexibility not present in traditional service compositions. If a player is not fulfilling the obligations set out in their contracts, a
number of measures can be taken; the player can be notified of the underperformance so they may rectify the issue, the contract can be altered to make an allowance for the underperformance, or as a final measure the player can be swapped for another at runtime. These actions are the responsibility of the entity playing the organiser role. If a role is absent a player at anytime due to the organiser reconfiguring the organisation the SMC can continue to operate for a time as messages sent to that role are buffered in a queue until a player is found, this also the case for the organiser role.

An SMC when viewed externally can be considered an autonomic element as it is capable of the required behaviours as defined by White (2004), these include self-management and establishing / maintaining relationships. ROAD also provides us a way to represent complex management relationships in organisational structures. Contracts between roles may stipulate a management relationship, e.g. supervisor-subordinate, peer-peer, buyer-seller, etc. Roles may also specify the differing levels of autonomy that is required of a player in order to play them. What is currently lacking in ROAD is the definition of a management interface for roles that allows the organiser to conduct negotiation (proposals and counter proposals) as well as maintenance on management relationships with external entities (referred to by White (2004) as negotiation and binding interfaces in the context of an autonomic element). These entities could potentially be other autonomic elements playing roles in the composition or more specifically in our case other ROAD SMC’s. Supporting mechanisms need to be in place to allow these external relationships to be established and then internalised as ROAD contracts. The work in this thesis seeks to address this issue.

4.2 Current ROAD implementation

The current implementation of ROAD exists as four tools or libraries. When used together they allow fully functioning ROAD composites to be instantiated and deployed into web service containers. Note that some of these tools are currently still in early development stages. The four tools / libraries are:

1. The **structural model** consists of a set of XML schemas that define the allowable structure and properties in an SMC. Using this model, a system designer may declaratively define SMC’s as XML documents which may then be instantiated by ROADfactory (described below). These XML files are both design and runtime artefacts. While a designer can manipulate the XML definitions to be consumed by
ROADfactory, SMC organisers also use these to communicate management information.

2. **ROADdesigner** is a visual modelling editor tool that allows the creation of SMC’s in a drag and drop fashion, eliminating the tedious problem of creating XML documents by hand. Once an SMC is defined using this tool, a valid XML document is exported which can be used by ROADfactory to instantiate a runtime version. ROADdesigner is also responsible for enforces validity of the SMC by checking generated XML documents against the schemas. ROADdesigner is being implemented as a plug-in for the Eclipse IDE.\(^\text{11}\)

3. **ROADfactory** is a library which is incorporated into other software projects that need to instantiate fully functioning SMC’s, but it may also be used as a standalone application. ROADfactory takes a supplied XML document (ideally generated from ROADdesigner) and produces a valid SMC Java object. It is this library which will be modified and extended in Chapter 7 to demonstrate the concepts discussed in this thesis.

4. **ROAD4WS** is an infrastructure with the purpose of allowing SMC’s to be deployed into a web service container, with all roles exposed as web services for players to interact with. ROAD4WS is currently under development as an extension of Axis 2.\(^\text{12}\)

### 4.3 Motivating example in the context of ROAD

To implement our motivating example described in the previous chapter requires a framework that is capable of representing systems with self-management capabilities (each cars system is a self-managed entity), creating agreements between those systems, and representing those agreements internally. Using ROAD each cars relationships can be represented by an SMC. Each SMC plays a role in the other when performing cooperative route planning and forming convoys. Figure 4.2 illustrates this and further incorporates the RTA as another SMC in which car SMC’s may play roles. In a real system many more roles

---


may be required to model the complete set of relationships the cars have with other entities. Here we focus only on the relationships between the SMC’s so we include a leader / follower car role in each to demonstrate internalising of external agreements i.e. roles such as follower / leader car are only bound by contracts to other convoy members and the RTA. Depending on the capability of the cars systems, either some intelligent agent or component may take on the role of the SMC’s organiser, or the drivers of the respective cars may do this. When in a convoy situation, the contracts between the car and convoy leader / follower roles contain terms for sending route and distance updates and obligations on those terms to enforce timing (e.g. send distance update every 10 seconds). The cars contracts with the RTA stipulates the RTA may send notifications of accidents, decreased weather conditions etc as described in Chapter 3. Currently ROAD does not describe a mechanism to allow a higher level SMC such as the RTA to put constraints on relationships between cars, i.e. distance constraints for convoys. With this in place more complex management structures such as mutual adjustment will become possible.
Chapter 4 An Introduction to the ROAD Framework

4.4 Summary

In this chapter we have given an overview of the ROAD framework, a meta-model for describing self-managed service compositions that may contain complex management relationships. In Section 4.1 we described the four essential components in a ROAD self-managed composite or SMC. In Section 4.2 we went on to briefly outline the current implementation of ROAD and identified the area we will be modifying / extending to demonstrate concepts described in this thesis. Section 4.3 illustrated our motivating example involving vehicle interactions in the context of ROAD SMC’s, and pointed out that ROAD currently lacks; mechanisms to allow management relationships with external parties to be created and then internalised as contracts, and mechanisms to allow SMC’s at higher levels of the organisational structure to enforce constraints on relationships between SMC’s at lower
levels. It is these problems this thesis seeks to address. In the next chapter we will describe the management communication types and operations required in order to allow SMC’s to establish complex management relationships with external parties.
5 Communication to Support Management Relationships

In this chapter we examine the communication requirements of self-managed systems wishing to enter into complex management relationships that extend beyond the traditional command-control approaches. As current standards and approaches are insufficient in facilitating the establishment of complex management relationships, we propose a new management interface that draws on different elements of existing standards. We then apply it in the web services domain. In Section 5.1 we examine the levels of management capability a system may possess in relation to its level of autonomy and how that affects the relationships that may be formed between systems with differing capabilities. This is followed by a discussion on the messaging requirements of each level in Section 5.2 In Section 5.3 we introduce the definition of the management interface that draws on aspects of multiple existing standards that is able to cater for the requirements discussed in Section 5.2 Finally we go on to illustrate how this management interface is incorporated into the ROAD framework in Section 5.4 and provide a summary in Section 5.5

5.1 Levels of management capability

As discussed in Chapter 2 a self-managed system may have differing levels of autonomy. This may entail a minimal level of self analysis / diagnostic capability that will reduce administration overhead or a high level of autonomy which may allow operation solely on high level organisational goals, without the need for human intervention. They may also exist across domains, have conflicting obligations, and have different technical capabilities. All these elements will have an impact on the kinds of relationships a system may establish with other parties. For example in our motivating scenario, if a car is not functionally capable of establishing management relationships, it may not be able to take part in the convoys of others. We will establish three simple levels or categories of runtime management capabilities a system may possess. These capabilities explicitly focus on an external view of the system in question, meaning they categorise the way in which external parties may interact. These are not a replacement, nor a competitor to the levels defined by others in
Chapter 5 Communication to Support Management Relationships

Chapter 2 such as the five levels defined by Colman and Han (2007); they are broad levels which can encompasses the more focused definitions as will be illustrated later. As this thesis is in the context of self managed service compositions, we will illustrate each level as a web service, although they are applicable to any distributed computing technology. The three levels of management capability are:

1. **No management capability.** Ordinary web services with no level of autonomy fall into this category. For example a web service that validates emails and provides a standard WSDL functional interface description. These services provide no means to establish agreements, alter their set functionality, or customise non functional aspects (such as QoS).

2. **Operational management capability.** We say a service that provides a way to configure the functionality it provides has operational management capability, i.e. the service provides a set type of functionality but it allows for configuration of the way it provides that functionality or the non functional requirements associated with it. In our scenario, the follower car sends distance updates every 10 seconds. If it allowed the leader car reconfigure this to 20 seconds via some exposed management operation, this is operational management. Other examples might be configuration of some form of system state or load balancing.

3. **Contractual capability.** The highest level on capability, contractual management refers to a service or system that can dynamically establish complex management relationships with other parties at runtime. This level encompasses those systems which are capable, at least at some level, of the forms of management relationships we have described thus far in this thesis. It is possible a service with this capability presents no functional interface publicly at all. Any functional operations exposed are done so on the basis of an established agreement with the external party.

There may also exist a sub level of capability relating to service binding. A service may only have operational management capability but may still have the ability to determine who may bind to it and make use of its functionality. We call this sub level binding autonomy and it is applicable to all three of the levels above.

The various levels of autonomy described in Chapter 2 by Colman and Han (2007) can be categorised into these broader levels as shown in Table 5.1.
Table 5.1: Management capabilities to levels of autonomy

<table>
<thead>
<tr>
<th>No management capability</th>
<th>No autonomy</th>
<th>Process autonomy</th>
<th>System state-autonomy</th>
<th>Intentional autonomy</th>
<th>Autonomy from constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational management capability</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Contractual capability</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Some of the levels of autonomy could potentially fit into different management capability categories depending on the external view of the system in question. A system with no autonomy may or may not, expose some operational capability that allows customisation of non functional aspects. Systems with process autonomy that can choose a process to meet some required system state may expose no external management capability. They may also expose some operational capability allowing an external party to have some impact on the choice of process. Systems with intentional autonomy and autonomy from constraints are independent and can choose whether not to comply with goals set forth by the organisation. This implies some form of agreement or management relationship is required in order for them to take part in the organisational structure. If this were not the case there can be no reasonable assurance they will fulfil the role in which they are placed, making the organisation unstable.

The level of management capability a system possesses will largely decide the management relationships it may enter into. A system with contractual management capability cannot form a complex management relationship with a system without management capability. In this case the situation is largely constrained to a command-control relationship where the system with a higher level of autonomy simply makes use of the other as a resource. The same applies to some extent to the systems with operational capability, although they may be configured at runtime by the resource consumer, making it more robust. The complex management patterns we wish to mimic are largely constrained to two parties with some level of contractual management capability but this does not mean systems with lesser capabilities cannot be used in an organisational structure. Take Figure 5.1 for example, which is an extension of Figure 2.2. A command-control relationship can be viewed as a highly constrained version of the direct supervision pattern, where the entity being supervised has no
option to opt out of, or alter the relationship. Therefore systems with lower capabilities are best suited to a position lower in the hierarchy in the operating core.

Figure 5.1: An organisation incorporating systems with lower management capabilities

Depending on the capability a system possesses, it may participate in management relationships that require a lesser capability. A system may deprecate its capability to suit an arrangement. For example a system with contractual management capability may enter into an agreement with another party that stipulates it must act as a system with operational management capability or perhaps no management capability at all. The difference is the system may opt out of the relationship at any time due to its independence and may still possess binding autonomy. An example of this in the context of our motivating scenario may be that a follower car in a convoy agrees to follow the leader car, send distance updates at the agreed interval, and allow the leader to modify the update interval at any time it chooses.

5.2 Messaging requirements

Each level of management capability has requirements for different forms of messages that may flow between it and others. If the message types can be identified, then systems can interact in a more standardised way with other parties. A system with no management capability has no manageability requirements. It simply needs to expose its functional
operations, e.g. in the case of a web service, via its WSDL interface. Operational and contractual capabilities are discussed in detail below.

5.2.1 Operational management messages

We previously said a system with operational management capability needs to expose some method in which its functionality can be customised by an external entity. In the web services domain, existing management interface standards discussed in Chapter 2 such as WS-Management or WSDM can allow this via the use of the following three basic message types as previously mentioned in Chapter 2.

- **Queries.** These provide the ability to query current configuration values such as system state, current load or in our motivating example the current send interval of distance updates from a following to a leader car.

- **Updates.** These allow an external party to configure the system. Many of the values that can be queried may also be updated, such as distance update send interval. Others may solely be query values such as current load.

- **Subscriptions.** Values should provide a subscription mechanism to allow asynchronous updates when values change such as system state.

These three patterns together allow an external party to make configuration changes, actively monitor a system (queries), and passively monitor a system (subscriptions).

5.2.2 Contractual management messages

Contractual management is based upon the presumption a system possesses some level of autonomy and needs to establish management relationships with other parties, in order to interoperate with the surrounding members of an organisation. With this in mind the following are the required message types (which draw on WS-Agreement (Andrieux et al., 2005).

- **Agreement discovery.** A system needs to be able to obtain a set of acceptable agreements from any other party it wishes to form a management relationship with. This provides a basis on which to build new agreement proposals and counter proposals.
Chapter 5 Communication to Support Management Relationships

- **Agreement formation.** The ability to send, receive, accept and reject proposals from other parties is required.

- **Agreement commencement.** Notifications that pre-defined agreement time or dates have been reached and some form of action is due to commence.

- **Agreement maintenance.** It may be required that existing relationships need to be altered and agreements renegotiated. There may also be a need for notifications of breaches of agreements.

- **Agreement termination.** Exiting or ending an agreement.

The message types described above must be flexible enough to allow a wide range of domain specific negation protocols and agreement formats.

### 5.3 A multi faceted management interface

Given the above messaging requirements for systems with various management capabilities, we now can now define a management interface that can present zero or more facets suitable to the type of management communication needed. Here we will apply this to the web services domain. In web services this could potentially be implemented in a number of ways, such as incorporating existing management standards for operational management. Here we will illustrate the management interface making use of WS-Management. The format of agreements and message content are left open as they are highly application domain specific. As we will explain in the next section, the ROAD framework uses XML representation of contracts for message content, which may be passed between parties instead of using standards like WS-Agreement.

Firstly we organise the facets of the management interface into port types, one for operational management and one for contractual. The operations inside these are summarised in Table 5.2.
Chapter 5 Communication to Support Management Relationships

Table 5.2: Management interface port types and operations

<table>
<thead>
<tr>
<th>Port types</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManOperational</td>
<td>Get</td>
</tr>
<tr>
<td></td>
<td>Set / Put</td>
</tr>
<tr>
<td></td>
<td>Subscribe</td>
</tr>
<tr>
<td>ManContractual</td>
<td>Propose</td>
</tr>
<tr>
<td></td>
<td>Reject</td>
</tr>
<tr>
<td></td>
<td>Accept</td>
</tr>
<tr>
<td></td>
<td>Terminate</td>
</tr>
<tr>
<td></td>
<td>Notify</td>
</tr>
</tbody>
</table>

The port types could potentially be separated into an entirely different endpoint as is done in other existing management standards. As is explained in the next section, in the ROAD framework we incorporate them into the existing functional interface. Below Figures 5.2 and 5.3 contain an example of the ManContractual and the ManOperational (using WS-Management standard compliant syntax) port types using some example ROAD specific message types.

```xml
<wSDL:portType name="ManContractual">
  <wSDL:operation name="GetAllowableAgreements">
    <wSDL:input message="road:AARequest" />
    <wSDL:output message="road:AAResponse" />
  </wSDL:operation>
  <wSDL:operation name="Propose">
    <wSDL:input message="road:agreement" />
  </wSDL:operation>
  <wSDL:operation name="Accept">
    <wSDL:input message="road:agreement" />
  </wSDL:operation>
  <wSDL:operation name="Reject">
    <wSDL:input message="road:agreement" />
  </wSDL:operation>
  <wSDL:operation name="Terminate">
    <wSDL:input message="road:agreement" />
  </wSDL:operation>
  <wSDL:operation name="Notify">
    <wSDL:input message="road:breach" />
  </wSDL:operation>
</wSDL:portType>
```

Figure 5.2: The ManContractual port type example
5.4 Incorporation into the ROAD framework

As described in Chapter 4, the ROAD framework is an organisational approach to creating self-managed service compositions. Recall that a ROAD Self-managed composite (SMC) consists of position descriptions called roles, which are bound by contracts that define the allowable interactions and obligations on those interactions between two role players. Also recall the organiser role is unique in that it exists one per SMC and allows an entity (whether it be human, or an intelligent agent) the ability to monitor and reconfigure the composition. That is the organiser can create, remove, and update agreements as well as altering the organisational structure. As it is the organiser’s responsibility to establish agreements with external parties and then internalise them as contracts, we need to allow management communication to flow from those external parties to the SMC’s organiser, and vice versa. To do this we will incorporate the concept of our multi faceted management interface described in Section 5.3.
Chapter 5 Communication to Support Management Relationships

The management interface will be added to the external functional interface already exposed by each role when an SMC is deployed in a web services container. Because of this we will consider players of roles in an SMC as either other ROAD SMC’s or web services. Depending on the requirements of a role, the management interface may also be expected to be exposed by a players required interface (recall that a role may have provided and required interfaces). With this in mind we can identify three possible scenarios involving the management interface and our levels of management capability:

1. A role in the SMC is being played by another contractually capable player. In this case the SMC organiser will need to communicate with the player to determine the terms of the agreement that will bind them and the obligations to one another. In this case both the SMC role and the player must expose a management interface with a contractual management facet (this holds whether the player is another ROAD SMC or a web service).

2. A role in the SMC is being played by a web service which is capable of operational management. In this case the player has no autonomy (at least none it is exposing to the outside world), meaning the SMC will not need to expose a management interface, as it assumed some level of intelligence is needed to make use of it on the players part.

3. A role is being played by a web service that has no management capability what so ever. The player has no ability to communicate management information to the SMC (just as a service with operational management capability). The player also exposes no management interface for the SMC to interact with.

The three scenarios are illustrated by a ROAD SMC with three roles (one per scenario) in Figure 5.4.
When put into the context of our motivating scenario we can see that as the leader and follower cars relationships are represented as ROAD SMC’s, they are both considered contractually capable players. This means they must both expose management interfaces with contractual management facets to allow their respective organisers to communicate as illustrated in Figure 5.5.
5.5 Summary

In this chapter we have explored the communication requirements of systems with differing levels of autonomy wishing to enter into complex management relationships with other parties. In Section 5.1 we discussed the different levels of management capability a system may possess and identified three levels; no management capability, operational management capability, and contractual management capability. We also explored how this restricts the management relationships systems at either level may enter into. In Section 5.2 we went on to look at the messaging requirements of each level of management capability. In Section 5.3 we introduced a multi faceted management interface, which is a management interface for the web services domain designed to cater for the requirements identified in Section 5.2. Finally in Section 5.4 we looked at how this management interface can be incorporated into the existing ROAD meta-model. In the next chapter we go on to look at abstract policy contracts in ROAD, which is a new type of contract that seeks to allow an SMC to model the mutual adjustment management pattern.
6 Policy Driven Management in ROAD

In this chapter we discuss abstract policy contracts in the ROAD framework. Abstract policy contracts are a new form of ROAD contract, which seek to allow ROAD SMC’s to structure the management of role players using the mutual adjustment management pattern discussed in Chapter 2. In Section 6.1 we look at the mutual adjustment management pattern in the context of the ROAD meta-model and the additional constructs which need to exist in order to allow it. In Section 6.2 we introduce ROAD abstract policy contracts in response to the requirements identified in Section 6.1. In Section 6.3 we go on to examine how abstract policy contracts can be used in the RTA SMC from our motivating scenario. Finally we give a summary in Section 6.4.

6.1 Mutual adjustment and ROAD

As we have seen, contracts in a ROAD SMC define the allowable interactions and obligations between two parties. In this way an SMC acts much like a form of middleware, any interactions between two role players are via their binding contract. When considered from the perspective of the organiser who is responsible for managing contracts and interacting with any management interfaces a player exposes, this is a form of the direct supervision management pattern as illustrated in Figure 6.1, which illustrates a scenario where the RTA SMC defines two convoy roles bound by a contract. The roles are played by car SMC’s and any communication directed from the leader car to the follower car is done so via the contract in the RTA SMC.

If an SMC were capable of the mutual adjustment management pattern, which is similar to policy based management approaches (discussed in Chapter 2) where policies are management decisions enforced on the interactions between lower level components of a system, then a significant amount of traffic through an SMC can be reduced and a layer of abstraction created. An SMC’s organiser could define a contract that needs to exist between two parties and then instruct those parties to create a relationship based on that contract. With this in place communication between those parties now flows only between them, not via the higher level SMC’s organiser. The SMC may also define only some high level details of the...
relationship and leave it to the role players to define specific realisation details; meaning the initial contract is only defined to the appropriate level of detail depending on what kind of layer of abstraction is required. When we apply this to the organisational structures discussed in Chapter 2, this type of management relationship is highly applicable at the top levels of an organisation, were managers make high level decisions / goals and instruct subordinates to take action to achieve those goals, without knowing the details of how they do so.

![Diagram of Direct Supervision in ROAD SMC's](image)

**Figure 6.1:** Direct supervision in ROAD SMC's

To achieve this in ROAD, an SMC’s organiser will require a number of constructs to be in place:

1. A new form of contract that defines the interactions and obligations at varying levels of detail between two parties. The contract does not require functional communication between the parties to flow via the SMC in which it resides (main point of difference with existing contracts). In this way the contract is implemented based on the high level constraints (or policies) imposed by the higher authority by the role players themselves. This is analogous to an abstract class being implemented by a derived class in object oriented programming where only some base details are inherited and the rest implemented as the situation dictates.

2. The ability to pass contract details to parties playing the roles bound by one of these abstract policy based contracts.
3. The ability to request updates on the status of the relationship between parties bound by an abstract based policy contract for management purposes.

Requirements 2 and 3 are possible via the multi faceted management interface discussed in Chapter 4 by proposing contract definitions, along with the details of the two parties that are to be involved in the contract (in regular contract negotiation the proposing party is considered one of the two parties). What remains then is the ability to define the abstract policy based contract itself in a ROAD SMC.

6.2 Abstract policy contracts in ROAD

As ROAD contracts are defined declaratively using XML schemas, we can define a new type of contract schema based on the existing one. We call this new form of contract an abstract policy contract for reasons mentioned in the previous section. An abstract policy contract does not actively have any functional or management communication pass through it; all functional communication is between the role players themselves bound by the abstract policy contract. Only management communication flows between the managing SMC and the subordinates, namely the transferral of the abstract policy contract details (in its ROAD XML representation so the role players may instantiate it) and any monitoring that may need to take place (e.g. status reports). An abstract policy contract can be defined at different levels of detail. What this means is an organiser may define an abstract policy contract containing the basic information on a relationship that needs to exist, for example some basic interaction permissions (or terms in ROAD speak) and obligations. The role players themselves must negotiate among themselves to fill out other necessary details of the contracts, e.g. more specific obligations on interactions, who will play which role, etc. An organiser may also define a very detailed abstract policy contract that the role players must realise if the situation requires. An example abstract policy contract in a generic SMC is illustrated in Figure 6.2. In this figure a manager defines an abstract relationship that must be realised by its subordinates in a way they see fit (as long as it still complies with the original abstract definition). The functional communication between the subordinates now flows between them exclusively instead of via the manager. By doing so management is now free to conduct other activities, confident that the subordinates are fulfilling goals imposed by management via the relationship imposed.
6.3 Abstract policy contracts and the Road Traffic Authority SMC

In our motivating scenario (described in detail in Chapter 3), the RTA wishes to impose policies on the way cars subscribed to its system interact (e.g. distance requirements) but may not wish to have the cars interact solely via its own system. By using abstract policy contracts in the way illustrated in Figure 6.2, the RTA may alter the relationships of the leader and follower cars as needed and is still free to send notifications on things such as accidents and dangerous road conditions. Cars now interact directly with one another as previously illustrated in Chapter 5, but with the added constraint of doing so under the guidance of the RTA. This is illustrated at a high level in Figure 6.3.
6.4 Summary

In this chapter we have described a new form of ROAD contract, namely an abstract policy contract, that allows the mutual adjustment management pattern to be modelled using ROAD SMC’s. In Section 6.1 we discussed the mutual adjustment management pattern in the context of ROAD SMC’s and defined some new requirements which will need to be fulfilled in order to allow it. In Section 6.2 we described the abstract policy contract which seeks to fulfil the requirements defined in Section 6.1. Finally in Section 6.3 we briefly discussed how an abstract policy contract might be used by the RTA SMC in our motivating scenario. In the next chapter we describe a prototype implementation built on the existing implementation of ROAD which demonstrates the concepts described in the previous chapters to be feasible.
Chapter 7 Implementation

7 Implementation

In this chapter we describe a prototype implementation demonstrating the concepts described in chapters 5 and 6 in the context of our motivating scenario from Chapter 3. The prototype seeks to demonstrate the use of management communication between ROAD SMC’s and also the use of abstract policy contracts which allow modelling of the mutual adjustment management pattern in ROAD. Section 7.1 gives a brief introduction to the current implementation of ROADfactory, the framework on which our prototype will be built. ROADfactory relies heavily on a number of open source frameworks, most notable of which is the Drools rules engine, for which we will provide an overview. In Section 7.2 we outline the extensions to ROADfactory allowing management communication to be conducted between ROAD SMC organisers (as described in Chapter 5). In Section 7.3 we go on to describe the ROADfactory extensions which demonstrate abstract policy contracts in the RTA SMC and describe how they are transferred to, and implemented by, the leader and follower car SMC’s. Finally we provide a summary in Section 7.4.

7.1 ROADfactory

As mentioned in Chapter 4, a goal of the ROAD framework is to allow SMC’s to be defined declaratively using XML. These SMC descriptions are then to be instantiated into fully functioning Java SMC objects. This description-to-instantiation process is the main purpose of the software library we call ROADfactory. ROADfactory is not the main concern of this chapter; however it is used extensively for our prototype implementation so we will provide a brief overview. The ROADfactory project in general not only encompasses instantiation of ROAD SMC’s, but also the implementation of the SMC’s themselves, including roles (functional and organiser), contracts and any other components required. ROADfactory relies on a number of open source libraries and Java API’s, the most notable of which are described briefly below.
ROADfactory uses JAXB\(^{13}\) (Java Architecture for XML Binding) to create binding classes whose instance variables map to the content of the SMC XML schemas (or structural model, mentioned in Chapter 4). SMC XML descriptions that comply with the schemas may then be used to create instances of the binding classes, which are then in turn used to populate the properties and configuration of an SMC object as illustrated in Figure 7.1. This creates a layer of abstraction between ROADfactory and the structure of the XML files. If the schemas are altered (which they frequently are considering the experimental nature of ROAD) the effect on ROADfactory is kept to a minimum as the bindings may be automatically regenerated.

![Figure 7.1: Definition-to-instantiation process in ROADfactory](image)

The Drools rules engine\(^{14}\) allows declarative definition of business rules which can be executed on demand using a Java API (a .Net version is also now available). The Drools engine has been incorporated into ROADfactory to allow the execution of rules which are coupled to the various components of a ROAD contract. As mentioned briefly in Chapter 4 ROAD contracts define the allowable interactions and obligations on those interactions between two roles. Contracts are made up of a number of complex components; terms (or interaction permissions), obligations, general clauses and conversation clauses. We will not

---


explore these in detail here as this has already been done by Singh (2008) and Colman (2007) yet it is important to note, that Drools rules may be executed by any component of a contract and are the mechanism for altering the overall state of a contract (allowable states consist of incipient, active (complying), active (breached), suspended, terminated and finalised). It should also be noted that while Drools has been chosen largely because of its active community and ease of use in Java, any rules engine could potentially be used in a ROAD SMC. To this end a suitable abstraction layer has been designed to decouple the implementation of contracts from the rules engine itself. Appendix A contains some example Drools rules.

The implementation of a ROAD SMC can be considered as two distinct layers as illustrated in Figure 7.2. The routing layer is responsible for ensuring the messages received by a role are sent to the appropriate destination role, via the appropriate contract. Rules may be defined which allow analysis on a messages content to determine the correct route; meaning an SMC could be considered a form of content based router. Underneath the routing layer we have the rules layer, which encompasses the execution of Drools rules based on the events generated by the routing layer (e.g. message received by contract, etc). The results of rule executions may affect the routing layer, such as messages being blocked due to contract states, context information and so on.

![Figure 7.2: Routing and rule layers in the SMC implementation](image)

SMC’s instantiated by ROADfactory are suitable for deployment into into a web services environment (among other environments if need be). This means each role will be exposed as a web service, including both its functional interface and management interface (as described in Chapter 5). Currently ROAD4WS, the web services SMC deployment environment, is still under development and therefore is not used in our prototype. The implementation has been
designed in such a way that the management related extensions will work readily in a web services environment, however we will be passing messages as Java objects, as a substitution for SOAP messages.

### 7.2 Management communication demonstration

In order to demonstrate the management communication concepts described in Chapter 5 we will extend ROADfactory. ROADfactory currently lacks the ability to pass management specific messages from a role player to an SMC’s organisar role. Currently any message received by a functional role will either be routed to another functional role or dropped if no valid routes are found. It is intended that when deployed in a ROAD4WS container, invocations on a role's web service interface will be captured and transformed into messages, or more specifically in our implementation, message wrappers. Message wrappers are tagged with the operation name that has been invoked. It is this operation name that is used to route a message to its appropriate destination. We can therefore check for known management message types such as propose, notify and accept / reject and route these messages directly to the SMC organisar role. On a functional role, message wrappers are passed in via put methods, which results in a message wrapper being placed in a queue and eventually being routed by worker threads when a worker is available. As this may cause a delay when routing management messages to the organisar role in times of high load, we create a separate put method for management message types, ensuring they can receive priority and be routed to the organisar as quickly as possible (which has its own queue to buffer management messages). This is illustrated in Figure 7.3.

![Diagram of management message routing](image-url)
Chapter 7 Implementation

To demonstrate our convoy motivating scenario from Chapter 3 we create an SMC XML definition (which is located in Appendix B), which contains the leader and follower car roles bound by a contract. The contract contains a term that allows distance updates to be sent from the follower role to the leader role. An obligation on this term defines a required performance of 5 seconds, meaning the follower car is obligated to send distance updates at this interval. The SMC definition does not define an organiser role as this is inherent in all SMC’s and does not need to be explicitly defined. This SMC definition can now be instantiated twice, once for the follower car and once for the leader car, as their required configurations are identical as can be seen by the illustrations in Chapter 3. The SMC’s can now be bound as role players in each other’s organisations.

Players of the various roles (leader car, follower car, leader car organiser and follower car organiser) are implemented as Java classes with pre-scripted actions. The implementation is illustrated by the class diagrams shown in Figure 7.4 (including the RtaOrganiser class used in the next section).

![Figure 7.4: UML class diagram of player implementations](image)

The outputs of the demonstration are log files (located in Appendix C) documenting each event that occurs as a result of the pre-scripted actions of the players, including non communication actions (i.e. altering contracts), as well as sending messages and receiving messages. Each individual event is time stamped and numbered so when log files are compared it can be deduced what has occurred. The following describes the major events that occur during the demonstration and indicate log file line numbers of the respective player’s
outputs where appropriate. The structure of the SMC’s and the role players in the demonstration are illustrated below in Figure 7.5.

**Figure 7.5:** Management communication demonstration SMC’s and players

1. FollowerCarPlayer begins sending distance updates at the required interval (every 5 seconds) to LeaderCarPlayer. This continues constantly throughout the demonstration.

2. FollowerCarOrganiser sends a *proposal* (correlation ID 1) to LeaderCarOrganiser for a new distance update interval of 60 seconds (line 1).

3. LeaderCarOrganiser receives *proposal* 1 and sends a *rejection* message to FollowerCarOrganiser (lines 1 and 2).

4. FollowerCarOrganiser receives *rejection* for proposal 1 (line 2).

5. FollowerCarOrganiser sends a *proposal* (correlation ID 2) to LeaderCarOrganiser for a new distance update interval of 20 seconds (line 3).

6. LeaderCarOrganiser receives *proposal* 2 and sends a *counter-proposal* message to FollowerCarOrganiser for an update interval of 10 (lines 3 and 4).
7. FollowerCarOrganiser receives *counter-proposal* 2 from LeaderCarOrganiser and responds with an *acceptance* message (lines 4 and 5).

8. FollowerCarOrganiser updates the required performance of the obligation in binding contract lc1-fc1 (line 6).

9. FollowerCarOrganiser sends an operational *set* message to FollowerCarPlayer to update its distance update send interval to 10 seconds (line 7).

10. LeaderCarOrganiser receives *acceptance* message 2 and updates its own lc1-fc1 contract (lines 5 and 6).

11. FollowerCarPlayer receives operational *set* message and updates its interval (line 8).

12. From this point onward FollowerCarPlayer sends distance update messages at the required 10 second interval.

We can see the above communications and actions demonstrate the management channel between SMC organisers allowing contractual management messages such as propose, accept, counter-propose, and reject. Operational messages are also demonstrated using a set message, sent from an SMC’s organiser to a player with operational management capability. The management communication between the two SMC’s results in alterations to an existing agreement, these alterations are then internalised as changes to the ROAD contracts. Once the contracts are altered to reflect the updated distance update interval, failure to meet the new required interval by FollowerCarPlayer will result in the contracts state changing (the severity of the state change is dependent on how far passed the required interval the distance updates are sent).

### 7.3 Abstract policy contract demonstration

In order to demonstrate the concept of an abstract policy contract as discussed in Chapter 6, we will further extend the prototype discussed in the previous section. ROADfactory currently does not contain the notion of a contract that does not allow functional or management messages to pass through it. The first step to achieving this is to add the ability to mark any given contract definition as abstract. As can be seen in the RTA SMC XML definition in Appendix B, a contract can be marked as abstract using the `<Abstract>` element. The result of this is that the terms (or interactions) defined in this contract are not placed into
an SMC’s routing table upon instantiation by ROADfactory. Any functional message sent to a role, bound by a contract marked as abstract, is not routed.

Management communication between players of roles connected by an abstract contract and the SMC’s organiser are still required, as this is the mechanism we will use to pass abstract contract definitions to players. An addition to the organiser role of an SMC is also required to allow an organiser to pass in an abstract contract definition and have it instantiated.

To demonstrate ROADfactory’s new functionality we will use a demonstration similar to that in the previous section, which makes use of players implemented in Java with pre-scripted actions. The player classes of the previous demonstration are used with the addition of an RTA SMC organiser player that proposes the use of an abstract policy contract to players of the roles bound by it (the leader and follower car SMC’s). In this demonstration three SMC XML definitions will be used (located in Appendix B) in a configuration much like the one illustrated in the convoy-RTA scenario in Chapter 3. Two definitions are for the leader and follower car SMC’s, each of which contains no contracts between roles. The SMC’s are also not playing roles in each other’s organisation as in the previous section. The third is an RTA SMC which contains roles for leader and follower cars bound by an abstract policy contract. The leader and follower car SMC’s are bound to the roles in the RTA SMC. As in the last demonstration the following describes the major events that occur during the demonstration and indicate log file line numbers of the respective player’s outputs (outputs located in Appendix C) where appropriate. The structure of the SMC’s and the role players in the demonstration are illustrated below in Figure 7.6.
Figure 7.6: Abstract policy contract demonstration SMC’s and players

1. RtaOrganiser sends a *proposal* (correlation ID 1) to SMC’s playing the leader and follower car roles. The proposal contains the definition of a convoy *abstract policy contract* (lines 1 and 2).

2. FollowerCarOrganiser and LeaderCarOrganiser receive the *proposal* sent by RtaOrganiser and respond with an *accept* message (line 1 and 2 in both players outputs).

3. RtaOrganiser receives *acceptance* messages from follower and leader car roles and assumes they have instantiated the *abstract policy contract*. 
Chapter 7 Implementation

4. FollowerCarOrganiser and LeaderCarOrganiser instantiate their respective copies of the *abstract policy contract* and bind to each other (play roles in each other’s organisation, as in the previous demonstration) (line 3 in both players’ outputs).

5. FollowerCarOrganiser sends an operational *set* message to FollowerCarPlayer to commence sending distance updates and to remain at a distance between 1 and 300 meters from the leader car (line 4).

6. FollowerCarPlayer receives the operational *set* message and begins sending distance updates (line 1).

7. RtaOrganiser makes a change to its *abstract policy contract* definition, changing the distance range between leader and follower to be between 200 and 300 meters (line 5) and proceeds to *propose* the change (correlation ID 2) to the leader and follower car roles (lines 6 and 7).

8. FollowerCarOrganiser and LeaderCarOrganiser receive *proposal* 2 and respond with an *accept* message (lines 5 and 6 for FollowerCarOrganiser, lines 4 and 5 for LeaderCarOrganiser). At this point the RTA assumes the change has been implemented.

9. FollowerCarOrganiser and LeaderCarOrganiser re-instantiate their respective copies of the *abstract policy contract* containing the update specified by the RTA (lines 7 and 6 respectively).

10. FollowerCarOrganiser sends an operational *set* message to FollowerCarPlayer to update its distance range value to be between 200 and 300 meters (line 8).

11. FollowerCarPlayer receives operational *set* and updates its distance range value (line 7).

12. From this point onward all distance updates from the follower car player to the leader car player contain a range between 200 and 300 meters.

From the outputs of the demonstration we can see the use of an abstract policy contract to create a relationship between two SMC’s that did not previously exist. The leader and follower car SMC’s implement the abstract policy contract proposed by the RTA and bind to each other. We can also see how a change in the abstract policy contract definition can be
realised by the SMC’s implementing it, by using the management communication types defined in Chapter 5.

7.4 Summary

In this chapter we have described a prototype implementation that seeks to demonstrate the concepts described in chapters 5 and 6 in the context of our convoy motivating scenario. Section 7.1 provided an overview of ROADfactory, the software library which instantiates SMC XML definitions into functioning runtime SMC’s implemented in Java. Section 7.1 also gave a brief introduction to some of the open source libraries ROADfactory relies upon. In Section 7.2 we went on to describe the prototype extensions to ROADfactory which allow SMC’s to exchange the management communication discussed in Chapter 5 and also explained the outputs generated by the prototype. Section 7.2 described the extensions to ROADfactory allowing abstract policy contracts (as discussed in Chapter 6) and again explained the outputs generated by the prototype. In the next chapter we will evaluate the work documented in this thesis and conclude.
8 Evaluation and Conclusion

In this chapter we conclude by summarising the contribution of this thesis, as well as evaluating how well the research questions discussed in Chapter 1 have been addressed. We also document the limitations of the work in this thesis and discuss possible future work. In Section 8.1 we discuss the contribution of this thesis in furthering the ability to structure self-managed systems into more complex management relationships, as well as its contribution to the ROAD framework in general. In Section 8.2 we discuss the works limitations and outline the possible future work. We summarise in Section 8.3.

8.1 Contribution

The goal of this thesis was to further the ability to structure self-managed software systems using complex management relationships in order to provide an increased level of flexibility. The approach taken was to examine existing management / organisational theory applied to human organisations to determine which patterns were most applicable to software systems. An existing framework, ROAD, was then extended to further its ability to allow structures based on the identified management patterns. The main contributions of this thesis are discussed below.

Management Communication

Organisational theory applied to human organisations was explored and patterns discussed as to their applicability to software systems. Two patterns were identified as being directly applicable; direct supervision and mutual adjustment defined by Mintzberg (1983). Direct supervision involves a manager closely directing and monitoring a subordinate(s), while mutual adjustment involves setting high level goals and allowing subordinates to coordinate among themselves to achieve them.

A standardised management communication model was explored in which systems with differing levels of autonomy are classified into three broad capability levels; no management capability, operational management capability and contractual management capability. System possessing operational management capability can allow external entities to configure
Chapter 8 Evaluation and Conclusion

non functional aspects of their operation, an ability allowed by some existing distributed management standards. A system possessing contractual management capability is considered to possess a high level of autonomy and cannot be managed by an external entity unless an agreement exists that dictates as such. The message requirements of each level of capability were identified as well as a discussion of how different levels of capability impact the types of relationships a system may enter into. A web services based management interface was defined with facets allowing for the messaging requirements of each capability level. Finally it was discussed how the levels of management capability as well as the management interface can be incorporated into ROAD, an existing meta-model for designing self-managed service compositions. When applied to ROAD, this management communication model as well as the multi faceted management interface, facilitates the ability for ROAD SMC’s to be structured into complex management relationships, including the direct supervision management pattern.

Abstract policy contracts

A new form of contract to be added to the ROAD meta-model was defined, called an abstract policy contract. An abstract policy contract seeks to allow ROAD SMC’s to be structured according to the mutual adjustment management pattern. By binding roles in a ROAD SMC with an abstract policy contract, the role players themselves have the responsibility of implementing the contract and establishing a relationship between themselves. This allows an SMC designer to create a layer of abstraction by only defining the elements of a contract which they see as necessary, leaving further details to be defined by the players. Functional communication flows only between players, not via the SMC containing the abstract policy contract.

Prototype implementation

The incorporation of the management communication model, as well as the abstract policy contract, has been demonstrated in a prototype demonstration that extends the current implementation of ROAD. The demonstrations are based upon a motivating scenario involving vehicles wishing to establish a convoy when travelling and a road traffic authority system wishing to impose constraints on the way cars communicate.
8.2 Limitations and future work

Selected management patterns

In this thesis only a subset of management patterns from human organisations are considered for application to ROAD. Additional patterns are identified in Chapter 2 which could potentially be explored further, such as standardised work process. Constructs to allow patterns such as this, will further the ability to allow ROAD SMC’s to model management patterns which may further increase flexibility and could be the focus of future work.

Web services deployment

While the multi faceted management interface was defined with web services concepts in mind, ROAD SMC’s have not yet been deployed into a web service environment. This is due to the deployment environment, ROAD4WS still being under development. In the interim the management communication model is demonstrated by passing messages between SMC’s using Java objects. Once development of ROAD4WS is complete, the web services definition of the management interface can be further evaluated.

Abstract policy contract lifecycle

The abstract policy contract concept has only been explored at a fundamental level. Outstanding issues to be addressed are; coordination issues between players implementing an abstract policy contract, i.e. how do they decide who will play what role of the contract and what are the details of the lifecycle of an abstract policy contract, a topic that has been explored in depth for the existing style of ROAD contracts. The ability for a supervisor SMC to impose monitoring constraints on players of roles bound by an abstract policy contract has also not been explored. These issues are left as future work.

Prototype implementation

The prototype implementation serves only to demonstrate concepts discussed in this thesis. Therefore further work is required to optimize some of the additions to the ROAD framework, most notably the abstract policy contract implementation. Players have also been implemented using pre-scripted actions, meaning the outputs of the prototype are not very flexible. Implementations of players that act according to flexible strategies instead of pre-scripted actions are considered out of the scope of this work.
8.3 Summary

In this chapter we have concluded by summarising the work in this thesis and highlighting its contribution to the area of management in self-managed systems, as well contributions to the ROAD framework. In section 8.1 we provided an overview of the previous chapters and noted how they address the research questions described in Chapter 1. In section 8.2 we went on to document the limitations of the work and noted areas to be explored further in future research.
Appendix A: Example Drools Rules

The following is an example of two Drools rules that could be used in the convoy scenario. The first rule is executed when the actual performance (time between distance updates) is within the range. The second is executed when the required performance has not been met. The result of either rule is a change in compliance level.

```drools
rule "distance-update-obligation-positive"
no-loop true
agenda-group "lc-fc.t1.ob1"
when
  obFact : ObligationFact( actualPerformance <= requiredPerformance)
then
  obFact.setComplianceLevel(0);
End

rule "distance-update-obligation-negative"
no-loop true
agenda-group "lc-fc.t1.ob1"
when
  obFact : ObligationFact( actualPerformance > requiredPerformance)
then
  obFact.setComplianceLevel(1);
End
```
Appendix B: ROAD SMC XML Definitions

Management communication demonstration SMC definition

The following is the SMC XML definition used in the management communication prototype.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:SMC name="Convoy SMC"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Roles>
    <Role id="lc1" name="Leader car">
      <Description>The leading car in a convoy</Description>
    </Role>
    <Role id="fc1" name="Follower car 1">
      <Description>A following car in a convoy</Description>
    </Role>
  </Roles>
  <Contracts>
    <Contract id="lc1-fc1" name="Leader car to follower car contract">
      <Abstract>False</Abstract>
      <State>Active</State>
      <Terms>
        <Term deonticType="permission" id="lc1-fc1.t1" messageType="message">
          <ObligatedParty>RoleB</ObligatedParty>
          <RequestSignature>distanceUpdateRequest</RequestSignature>
          <ResponseSignature>distanceUpdate</ResponseSignature>
          <Obligations>
            <Obligation id="lc1-fc1.t1.ob1" name="term 1 obligation 1" type="temporal">
              <RequiredPerformance>5</RequiredPerformance>
              <Description>Obligated party must send distance updates at the specified interval</Description>
            </Obligation>
          </Obligations>
          <Description>Allows distance updates from a follower car to a leader car</Description>
        </Term>
      </Terms>
      <GeneralClauses>
    </GeneralClauses>
  </Contracts>
</tns:SMC>
```
Appendix B: ROAD SMC XML Definitions

```xml
</GeneralClauses>
    <ConversationClauses>
    </ConversationClauses>
<RoleAID>1cl</RoleAID>
<RoleBID>fcl</RoleBID>
<Description>A contract between a leader and follower car in a convoy</Description>
</Contract>
</Contracts>
<Description>An SMC for a convoy with a leader car and potentially many follower cars</Description>
</tns:SMC>
```
Appendix B: ROAD SMC XML Definitions

Abstract policy contract demonstration SMC definitions

The following SMC XML definitions are used in the prototype to demonstrate abstract policy contracts.

RTA SMC

```xml
<?xml version="1.0" encoding="UTF-8"?
<tns:SMC name="RTA SMC"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Roles>
    <Role id="lc1" name="Leader car">
      <Description>The leading car in a convoy</Description>
    </Role>
    <Role id="fc1" name="Follower car 1">
      <Description>A following car in a convoy</Description>
    </Role>
    <Role id="rta" name="RTA">
      <Description>The RTA</Description>
    </Role>
  </Roles>
  <Contracts>
    <Contract id="lc1-fc1" name="Leader car to follower car contract">
      <Abstract>True</Abstract>
      <State>Active</State>
      <Terms>
        <Term deonticType="permission" id="lc1-fcl.t1" messageType="message">
          <ObligatedParty>RoleB</ObligatedParty>
          <RequestSignature>distanceUpdateRequest</RequestSignature>
          <ResponseSignature>distanceUpdate</ResponseSignature>
          <Obligations>
            <Obligation id="lc1-fc1.t1.ob1" name="term 1 obligation 1" type="temporal">
              <RequiredPerformance>5</RequiredPerformance>
              <Description>Obligated party must send distance updates at the specified interval</Description>
            </Obligation>
            <Obligation id="lc1-fc1.t1.ob2" name="term 1 obligation 2" type="non-temporal">
              <RequiredPerformance>1-300</RequiredPerformance>
              <Description>Obligated party stay within the specified distance parameters</Description>
            </Obligation>
          </Obligations>
        </Term>
      </Terms>
    </Contract>
  </Contracts>
</tns:SMC>
```
Appendix B: ROAD SMC XML Definitions

<Description>Allows distance updates from a follower car to a leader car</Description>

</Term>
</Terms>

<GeneralClauses>
</GeneralClauses>

<ConversationClauses>
</ConversationClauses>

<RoleAID>lc1</RoleAID>
<RoleBID>fc1</RoleBID>

<Description>A contract between a leader and follower car in a convoy</Description>

</Contract>

<Contract id="rta-lc1" name="RTA to leader car contract">
  <Abstract>False</Abstract>
  <State>Active</State>
  <Terms>
    <Term deonticType="permission" id="rta-lc1.t1" messageType="message">
      <ObligatedParty>RoleB</ObligatedParty>
      <RequestSignature>weatherUpdateRequest</RequestSignature>
      <ResponseSignature>weatherUpdate</ResponseSignature>
      <Obligations>
      </Obligations>
      <Description>Allows weather updates from the RTA to the leader car</Description>
    </Term>
  </Terms>
</Contract>

<Contract id="rta-fc1" name="RTA to follower car contract">
  <Abstract>False</Abstract>
  <State>Active</State>
  <Terms>
    <Term deonticType="permission" id="rta-fc1.t1" messageType="message">
      <ObligatedParty>RoleB</ObligatedParty>
      <RequestSignature>weatherUpdateRequest</RequestSignature>
      <ResponseSignature>weatherUpdate</ResponseSignature>
      <Obligations>
      </Obligations>
      <Description>Allows weather updates from the RTA to the follower car</Description>
    </Term>
  </Terms>
</Contract>
Appendix B: ROAD SMC XML Definitions

Leader car SMC

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:SMC name="Leader car convoy SMC"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

<Roles>
  <Role id="lc1" name="Leader car">
    <Description>The leading car in a convoy</Description>
  </Role>
  <Role id="fc1" name="Follower car 1">
    <Description>A following car in a convoy</Description>
  </Role>
  <Role id="rta" name="RTA">
    <Description>The RTA</Description>
  </Role>
</Roles>

<Contracts>
  <Contract id="rta-lc1" name="RTA to leader car contract">
    <Abstract>False</Abstract>
    <State>Active</State>
    <Terms>
      <Term deonticType="permission" id="rta-lc1.t1" messageType="message">
        <ObligatedParty>RoleB</ObligatedParty>
        <RequestSignature>weatherUpdateRequest</RequestSignature>
        <ResponseSignature>weatherUpdate</ResponseSignature>
        <Obligations>
          <Description>Allows weather updates from the RTA to the leader car</Description>
        </Obligations>
      </Term>
    </Terms>
  </Contract>
</Contracts>
</tns:SMC>
```
Appendix B: ROAD SMC XML Definitions

Follower car SMC

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tns:SMC name="Follower car convoy SMC"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <Roles>
    <Role id="lc1" name="Leader car">
      <Description>The leading car in a convoy</Description>
    </Role>
    <Role id="fc1" name="Follower car 1">
      <Description>A following car in a convoy</Description>
    </Role>
    <Role id="rta" name="RTA">
      <Description>The RTA</Description>
    </Role>
  </Roles>
  <Contracts>
    <Contract id="rta-fc1" name="RTA to follower car contract">
      <Abstract>False</Abstract>
      <State>Active</State>
      <Terms>
        <Term deonticType="permission" id="rta-fc1.t1" messageType="message">
          <ObligatedParty>RoleB</ObligatedParty>
          <RequestSignature>weatherUpdateRequest</RequestSignature>
          <ResponseSignature>weatherUpdate</ResponseSignature>
          <Obligations/>
          <Description>Allows weather updates from the RTA to the follower car</Description>
        </Term>
      </Terms>
    </Contract>
  </Contracts>
  <Description>An SMC for a convoy with a leader car and potentially many follower cars</Description>
</tns:SMC>
```
Appendix B: ROAD SMC XML Definitions

</tns:SMC>
Appendix C: Prototype Outputs

Management communication demonstration outputs

The following are the logged outputs from each player in the management communication demonstration prototype.

Leader car organiser output

<table>
<thead>
<tr>
<th>Time</th>
<th>Message Type</th>
<th>ID</th>
<th>Destination</th>
<th>Sender</th>
<th>New distance update interval proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:12:04</td>
<td>Proposal</td>
<td>1</td>
<td>Role fc1</td>
<td>Role fc1</td>
<td>60 seconds</td>
</tr>
<tr>
<td>14:12:06</td>
<td>Rejection</td>
<td>1</td>
<td>Role fc1</td>
<td>Role fc1</td>
<td></td>
</tr>
<tr>
<td>14:12:09</td>
<td>Proposal</td>
<td>2</td>
<td>Role fc1</td>
<td>Role fc1</td>
<td>20 seconds</td>
</tr>
<tr>
<td>14:12:11</td>
<td>Counter-Proposal</td>
<td>2</td>
<td>Role fc1</td>
<td>Role fc1</td>
<td>10 seconds</td>
</tr>
<tr>
<td>14:12:14</td>
<td>Acceptance</td>
<td>2</td>
<td>Role fc1</td>
<td>Role fc1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACTION</td>
<td></td>
<td></td>
<td></td>
<td>Updating contract lc1-fc1 Obligation lc1-fc1.t1.ob1. New required performance: 10</td>
</tr>
</tbody>
</table>

Follower car organiser output

<table>
<thead>
<tr>
<th>Time</th>
<th>Message Type</th>
<th>ID</th>
<th>Destination</th>
<th>Sender</th>
<th>New interval proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:12:04</td>
<td>Proposal</td>
<td>1</td>
<td>Role lc1</td>
<td>Role lc1</td>
<td>60 seconds</td>
</tr>
<tr>
<td>14:12:06</td>
<td>Rejection</td>
<td>1</td>
<td>Role lc1</td>
<td>Role fc1</td>
<td></td>
</tr>
<tr>
<td>14:12:09</td>
<td>Proposal</td>
<td>2</td>
<td>Role lc1</td>
<td>Role fc1</td>
<td>20 seconds</td>
</tr>
<tr>
<td>14:12:11</td>
<td>Counter-Proposal</td>
<td>2</td>
<td>Role lc1</td>
<td>Role fc1</td>
<td>10 seconds</td>
</tr>
<tr>
<td>14:12:14</td>
<td>Acceptance</td>
<td>2</td>
<td>Role lc1</td>
<td>Role fc1</td>
<td></td>
</tr>
</tbody>
</table>
Appendix C: Prototype Outputs

6. Time: 14:12:14. ACTION :: Updating contract lc1-fc1 Obligation lc1-fc1.t1.ob1 New required performance 10


Leader car output

1. Time: 14:11:44. RECEIVED :: Message type: Distance update. Sender: Role fc1. Current distance: 210m


Follower car output

### Appendix C: Prototype Outputs

<table>
<thead>
<tr>
<th>Time</th>
<th>Message Type</th>
<th>Destination</th>
<th>Current Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:11:49</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>150m</td>
</tr>
<tr>
<td>14:11:54</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>99m</td>
</tr>
<tr>
<td>14:11:59</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>103m</td>
</tr>
<tr>
<td>14:12:04</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>66m</td>
</tr>
<tr>
<td>14:12:09</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>133m</td>
</tr>
<tr>
<td>14:12:14</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>183m</td>
</tr>
<tr>
<td>14:12:17</td>
<td>Set</td>
<td>Role fcOrg</td>
<td>New distance update interval: 10 seconds</td>
</tr>
<tr>
<td>14:12:19</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>160m</td>
</tr>
<tr>
<td>14:12:29</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>126m</td>
</tr>
<tr>
<td>14:12:39</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>267m</td>
</tr>
<tr>
<td>14:12:49</td>
<td>Distance update</td>
<td>Role lc1</td>
<td>132m</td>
</tr>
</tbody>
</table>
Appendix C: Prototype Outputs

Abstract policy contract demonstration outputs

The following are the logged outputs from each player in the abstract policy contract demonstration prototype.

RTA organiser output


Leader car organiser output


2. Time: 14:10:47. SENDING :: Message type: Acceptance. ID: 1. Destination: Role rta

3. Time: 14:10:47. ACTION :: Instantiating abstract policy contract lcl-fcl

Appendix C: Prototype Outputs


Follower car organiser output


2. Time: 14:10:47. SENDING :: Message type: Acceptance. ID: 1. Destination: Role rta

3. Time: 14:10:47. ACTION :: Instantiating abstract policy contract lcl-fcl


Leader car output


2. Time: 14:10:55. RECEIVED :: Message type: Distance update. Sender: Role fcl. Current distance: 64m

3. Time: 14:11:00. RECEIVED :: Message type: Distance update. Sender: Role fcl. Current distance: 39m


Appendix C: Prototype Outputs


Follower car output


References


References


Colman, A. & Han, J. (2005a). Operational management contracts for adaptive software organisation. In Proceedings of the Australian Software Engineering Conference (ASWEC’05), Brisbane, Australia, ACM.


Colman, A. & Han, J. (2005c) Organizational roles and players. Roles, an Interdisciplinary Perspective: Ontologies, Programming Languages, and Multiagent Systems: Papers from the AAAI Fall Symposium.


References


