Formalised Responsibility Modelling for Automated Socio-Technical Systems Analysis

Robbie Simpson

Submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy

School of Computing Science

College of Science and Engineering
University of Glasgow

October 2017

Abstract

Modelling the structure of social-technical systems as a basis for informing software system design is a difficult compromise. Formal methods struggle to capture the scale and complexity of the heterogeneous organisations that use technical systems. Conversely, informal approaches lack the rigour needed to inform the software design and construction process or enable automated analysis.

We revisit the concept of *responsibility modelling*, which models social technical systems as a collection of actors who discharge their responsibilities, whilst using and producing resources in the process. In this thesis responsibility modelling is formalised as a structured approach for socio-technical system specification and modelling, with well-defined semantics and support for automated structure and validity analysis.

We provide structured definitions for entity types and relations, and define the semantics of delegation and dependency. A constraint logic is introduced, providing simple specification of complex interactions between entities. Additionally, we introduce the ability to explicitly model uncertainty. To support this formalism, we present a new software toolkit that supports modelling and automatic analysis of responsibility models in both graphical and textual form.

The new methodology is validated by applying it to case studies across different problem domains. A study of nuclear power station emergency planning is validated by comparison to a similar study performed with earlier forms of responsibility modelling, and a study of the TCAS mid-air collision avoidance system is validated by evaluation with domain experts. Additionally, we perform an explorative study of responsibility modelling understanding and applicability through a qualitative study of modellers.

Acknowledgements

Supervisorial thanks to Tim Storer for his advice, guidance and employment offers over the course of this PHD; also to Karen Renaud, even if this research didn't quite turn out how we expected at the start.

Thanks to all my colleagues in the Systems Engineering subsection for their support over the years, participation in experimental studies and for helping to ensure a sufficient supply of biscuits. Thanks also to the teaching office, tutors and students I've worked with over the many years; teaching does keep you sane at times.

Parts of this research was carried out in conjunction with Sophrodyne Ltd. and supported by a Scottish Funding Council Innovation voucher. Thanks to them for their involvement, and to SFC for the money.

Ultimate thanks should go to my parents for their multi-faceted support over the duration of this PHD, without which it would never have been completed.

Contents

List of Figures

Chapter 1

Introduction

> "It is very important to use a modelling language which is well-understood, both by its authors and by its users. This points towards the disciplined use of small, expressive, languages that have a formal semantics, that are implemented with a high-degree of integrity, and which employ constructs that naturally support the modelling idiom." Collinson et al. [32]

1.1 Motivation

Social-technical systems models which capture technical, human and organisational concerns are recognised as an important element of software development [11], but modelling and analysing them is often a difficult compromise. These systems involve complex interactions between humans, machines and the environment around the system. They cannot be treated purely as technical problems or as human factors issues, as complex interactions between technical elements and human elements are vital to their correct understanding [11]. Such systems generally feature multiple system goals and multiple ways for achieving these goals; the design and implementation of socio-technical systems requires techniques that can effectively balance competing goals while accurately modelling both human and technical elements.

Large-scale social-technical systems are becoming increasingly prevalent as purely manual or paper-based processes are updated to take advantage of computing capabilities. Many of these projects suffer embarrassing delays, cost overruns or complete failure due to poor requirements specification. Other systems are completed but fail to provide key functionality or fail to integrate with existing operational procedures. For example, consider the United States's 'HealthCare.gov' integrated healthcare system project, which failed to integrate a range of state-level, Federal and private-sector organisations [30] or the TAURUS trading

system for the London Stock Exchange, which at one point consisted of seventeen different system designs [46].

In addition, the complexity of these systems can often lead to serious and unexpected failures. While these failures can ultimately be traced to poor specification or understanding of the requirements, a system may appear to be completed successfully according to well-specified requirements, only to later fail in stressed conditions not envisioned by the system designers. For example, the crash of Air France Flight 447 in difficult weather conditions can be considered a failure of the combined socio-technical system of the pilots and the aircraft's fly-by-wire system, rather than as an individual failure by the pilots or the technical system alone [153].

Formal approaches to requirements elicitation and system modelling such as Z [144] or Communicating Sequential Processes (CSP) [78] can provide many useful benefits, such as a clear and unambiguous syntax and a wide range of tool support such as model checking or vulnerability analysis. However, their choices of formalism rarely include concepts that specifically address human and organisational concerns, so modelling social elements is difficult, unnecessarily complex and sometimes impossible.

Alternatively, less formal methodologies can be used, such as Soft Systems Methodology [26] or i* [203]. Approaches such as goal-oriented modelling or systems-theoretic techniques allow reasoning over the full scope of social-technical systems, capturing both social and technical issues. These approaches can however cause problems at the development stage, as they are open to subjective interpretation. Comparisons between different system versions or levels of abstraction are also difficult, as the ambiguity inherent to the approach can lead to inconsistency.

Responsibility modelling is an informal approach to requirements gathering and systems analysis built around the concept of responsibilities that should be discharged. These responsibilities are assigned to actors and may depend upon or produce resources. Proponents of responsibility modelling argue that responsibilities are a more natural abstraction of system behaviour than goals or tasks [40]. Additionally, the concept of obligations (upon actors) provides a richer model of user behaviour compared to similar goal-based approaches.

However, responsibility modelling remains a semi-structured approach. There is no consistent notation, formal rules for transformation of models or well defined semantics over complex issues such as delegation of responsibilities. This makes providing effective tool support difficult and prevents consistent application of the technique.

These limitations make the effective use of responsibility modelling difficult. Inconsistent notations make understanding models difficult, and the unclear semantics of important features makes them impossible to analyse consistently. However, responsibility modelling's strengths such as its wide range of abstraction, intuitive underlying concepts and breadth of application

could prove effective if its current shortcomings are addressed.

1.2 Thesis Statement

In order to provide effective and structured analysis of socio-technical systems, responsibility modelling can be formalised with well-defined semantics that retain its abstractive power and ease-of-use, enabling significant amounts of model analysis to be automated and allowing new forms of analysis.

A formalised variant of responsibility modelling can act as an effective compromise, combining much of the useful rigour of formal methods while maintaining the scope and flexibility of usage of more informal approaches. Effective tool support becomes possible, allowing for a range of automatic analyses such as completeness, validity and overload checking. Models can be created at multiple levels of abstraction and can be directly related by model transformations and discharge checking. Ambiguity is reduced, especially over complex actor relationships, which allows the technique to be more consistently applied using a well-defined process.

1.3 Research Questions

In order to fully elaborate formalised responsibility modelling we propose the following research questions:

Q1 - Can responsibility modelling be formalised in a consistent manner?

Q2 - Can automatic analysis techniques and tooling be developed for formalised responsibility modelling?

Q3 - Can formalised responsibility modelling be applied consistently and without specialist knowledge?

Q4 - Does formalised responsibility modelling provide easier or more detailed analysis than previous responsibility modelling approaches?

Q5 - Does the application of formalised responsibility modelling produce models and results that are comparable to domain-specific approaches or expert analysis?

Question 1 addresses the core of the thesis - is it possible for responsibility modelling to be defined in a consistent, well-structured manner? A more formal definition can add valuable rigour, but poorly constructed formalisms can limit the flexibility of a technique or lead to unresolvable inconsistencies. We will show that such a formalisation is possible while maintaining the key advantages of the responsibility modelling approach.

Question 2 emphasises one of the core claims of the thesis - that the formalisation of responsibility modelling will allow for automated analysis of models, and a key part of this is effective tool support. We will define multiple forms of analysis that can be completely or partly automated, and present a toolkit that enables modelling and analysis using these techniques and evaluate their efficacy in comprehending socio-technical system structure.

Question 3 covers the reproducibility of the technique. Many social-technical modelling and analysis techniques are highly subjective, with the results of the analysis dependent on the skill and background of the analyst. By defining modelling and analysis processes we aim to show that this technique can be applied with a significantly lower level of subjectivity.

Question 4 addresses the value of formalised responsibility modelling as an analysis technique. The value of responsibility modelling is already well argued in the literature, so we show that formalisation extends the technique with additional levels of analysis, while maintaining the strengths of traditional responsibility modelling. In particular, we demonstrate the value of automated analysis, which enables practical analysis of larger models and increases the consistency of results.

Question 5 considers a more ambitious claim - that the results of responsibility modelling (a general-purpose modelling technique) can be broadly comparable to the results of expert analysis or the application of specialised, domain specific techniques. Proving or disproving such a claim in general is impractical due to the range of potential techniques and domains, but we will examine the application of responsibility modelling in several problem areas and draw limited conclusions.

1.4 Research Contributions

Important contributions to the field of research in this thesis include:

A methodology for iterative modelling and evaluation of case studies

Many case studies of socio-technical modelling methods struggle to demonstrate effective validation of their models, which potentially undermines confidence in the results and implications from these studies. We present a new iterative modelling and evaluation methodology that draws on action research and design science to provide an effective process for iteratively developing models while evaluating them against different data sources.

Formal definitions of core responsibility modelling concepts

In previous research the set of responsibility modelling entities and relationships has varied substantially from paper to paper, in both the types used and the meaning of each type. We select a minimal set of basic entities and relations that retains the full expressiveness of existing approaches, while simplifying the range required. We provide concrete definitions for

each of these, and demonstrate how they can be used to construct more complex interactions. This is accompanied by a methodological discussion of difficult modelling issues, such as levels of abstraction and scope.

Extensions to existing responsibility modelling notation

With the core primitives defined it becomes possible to develop modular extensions to the notation with additional features and functionality. We define a satisfaction logic that allows for the specification of complex relations between entities, incorporating common socio-technical concepts such as fallback modes and redundancy. Support is added for explicitly modelling uncertainty and indicating where lack of information may limit analysis. Formally defined transformations are provided for common model changes such as the delegation of responsibilities.

Tool support for construction and analysis of responsibility models

Introducing formally defined semantics to responsibility modelling allows for substantial automated analysis, but this requires proper tool support. We have developed a software toolkit that supports graphical creation and editing of responsibility models, and provides automated analysis on these models. These analysis techniques include criticality detection, overload analysis and reliance checks, enabling broader and more consistent analysis of responsibility models.

A comparative case study of formalised and unformalised responsibility modelling

We provide direct comparison of formalised responsibility modelling against an early version of the technique by re-examining a case study of the Hunterston Nuclear Power Stations Off-Site Emergency Plan, which had previously been studied with responsibility modelling by the InDeED consortium. Our analysis of this case study identifies the same core issues as the previous study, but additionally detects several potential vulnerabilities that were not previously detected.

Application and evaluation of responsibility modelling in the aviation domain

We perform a study on the use of responsibility modelling in aviation by modelling, analysing and evaluating the TCAS aircraft collision warning system using responsibility modelling and evaluating these results with domain experts. This provides two main contributions; firstly, demonstrating the use of responsibility modelling in a new domain; secondly, evaluating the correctness, understandability and relevance of responsibility modelling to experts in a particular domain.

Qualitative evaluation of responsibility usage by non-expert users

We conduct a user study asking participants to construct a responsibility model of a simple socio-technical system, and analyse the resulting models and evaluate participants' use and understanding of responsibility modelling. This study identifies responsibility modelling

concepts found difficult by users and examines common characteristics in the application of responsibility modelling. Additionally, users' recommendations for extensions to responsibility modelling are presented and analysed. This is (to the best of our knowledge) the first evaluation of responsibility modelling involving non-expert users.

1.5 Thesis Structure

This dissertation continues in Chapter 2 with a survey of related work in social-technical systems modelling and analysis, including both formal and informal approaches and earlier versions of responsibility modelling. This includes a range of techniques from the systems modelling, requirements engineering and safety analysis fields.

Chapter 3 discusses modelling and evaluation methodologies in socio-technical systems research, and proposes a new case-study based modelling and evaluation methodology based on a synthesis of existing work. This methodology provides a structured approach to constructing models based on iterative rounds of modelling and evaluation using different sources of information.

The core concepts and notation of formalised responsibility modelling are then introduced in Chapter 4, which provides well-defined semantics and logics for responsibility modelling entities and relationships. In addition, it defines several extensions to the previous state-of-the-art in responsibility modelling, including explicit uncertainty and a satisfaction logic for expressing complex relationships. Important modelling issues are also discussed, and the formalised notation is demonstrated in several worked examples.

This is followed by Chapter 5, which introduces several semi-formal analysis methodologies and discusses automated analysis techniques. In addition, it presents new tool support for constructing and analysing formalised responsibility models. A series of automated analysis methods are defined and their applications discussed, and semi-formal techniques are explored to provide broader analysis.

Two cases studies follow, showing the strengths and weaknesses of formalised responsibility management in different settings. Chapter 6 provides a comparison to previous responsibility modelling techniques by re-examining the Hunterston off-site emergency plan previously analysed by the InDeED project. Chapter 7 applies responsibility modelling to the TCAS air collision avoidance system, and assesses the conclusions reached and the understandability of responsibility modelling through evaluation with domain experts.

Chapter 8 describes a user study to examine the use of responsibility modelling by non-expert modellers. Experimental participants constructed a responsibility model of a simple socio-technical system, and their understanding, application and experiences using the technique

are analysed. Finally, overall conclusions and suggestions for future research appear in Chapter 9.

Chapter 2

Survey

2.1 Introduction

This chapter provides a comprehensive overview of relevant research in socio-technical modelling and analysis. This includes material from system modelling, safety analysis, software engineering and formal logics, representing the wide theoretical and practical underpinnings of socio-technical techniques.

As a result, this survey includes material from a wide range of different backgrounds, and from the early origins of software and systems engineering to the present day. A broad overview of each technique or concept is provided, with a particular focus on the published demonstration and evaluation of techniques via examples or case studies. Additionally, the functionality of tool support for these methodologies is considered where such tooling exists.

Firstly, the chapter begins by examining the origins of the term 'socio-technical system' and the underlying philosophy of socio-technical systems design, and discusses the relationship between the narrow field of socio-technical research and the broad range of methods that can be applied to socio-technical problems. This provides the context and terminology in which the remainder of the survey is situated.

The chapter continues by considering early software development and the emergence of requirements engineering as a field of research, as well as early requirements engineering techniques. This provides an understanding of the core issues in the modelling of software and socio-technical systems and shows the roots of modern methods. This is followed by a brief examination of formal methods and their suitability for modelling socio-technical systems. The complexity of such methods often renders them unsuitable for wide-scale use, but their power and rigour highlight the advantages of more structured modelling. Recent requirements engineering techniques such as goal-oriented and agent-based modelling techniques are then covered, representing the current standard in requirements-driven system modelling.

Technique	*Era*	*Core Concept*	*Applications*
MORT	1970s	Risk Trees	Safety Analysis
HAZOPs	1970s	Process Deviation	Risk Analysis
SREM	1970s	-	Requirements Engineering
SSADM	1980s	-	Systems Analysis & Design
Z	1980s	Set Theory	Formal Specification
Soft Systems Methodology	1980s	Holistic Systems Theory	Process Improvement, Systems Analysis
i*	1990s	Agent-based modelling	Requirements Engineering, Process Improvement
KAOS	1990s	Goal-based modelling	Requirements Engineering
BPMN	2000s	Data Flow	Process Improvement, Systems Design
SysML	2000s	Systems-of-Systems	System Requirements & Design
STAMP	2000s	Systems Theory	Safety Analysis
Responsibility Modelling	2000s	Responsibilities	Requirements Engineering, System Analysis
Timebands	2000s	Variable Timing	Performance Timing
FRAM	2010s	Functional Resonance	Safety Analysis

Figure 2.1: Overview of techniques examined

We move on to consider business process modelling and system-of-systems modelling; both demonstrate techniques for modelling large, real-world systems with a strong focus on socio-technical components. We also examine several structured safety analysis techniques, which provide tools for modelling and analysing complex systems with a particular consideration of human elements.

Soft Systems Methodology is examined as an exemplar of holistic systems thinking approaches, and deontic logic and the Timebands approach are investigated as suitable formalisations for modelling obligations and temporal behaviour respectively.

Finally, the chapter ends by exploring the full history of responsibility modelling, beginning with the use of responsibilities as a concept for requirements elicitation before covering graphical techniques for modelling individual responsibilities. The chapter concludes with a comprehensive examination of responsibility modelling as a technique for the specification and analysis of socio-technical systems. An overview of the main techniques covered is shown in Figure 2.1.

2.2 Socio-technical systems

Baxter and Sommerville [11] define a socio-technical system as ‘describ[ing] systems that involve a complex interaction between humans, machines and the environmental aspects of the work system’. This draws on the origins of the term ‘socio-technical’ in early work by Emery and Trist, starting in the 1950s. Trist [184] provides an overview of their early work, which began at the Tavistock Institute. Researchers there performed action research investigations of working practices in coal mines, and examined how changes in technology and machinery had led to the elimination of social collaboration in the working environment, but which had recently been reintroduced along with new equipment. Inspired by this they sought to develop

a theory of work organisations that combined these social and technological factors - socio-technical systems theory. Trist's focus is explicitly on work organisations, and specifically the self-organisation of workers and the relationship between workers and management in blue-collar occupations. This holistic view of work structure achieved international success through the 1960s and 1970s, but began to decline in the 1980s due to changes in the business climate [129]. Instead, several socio-technical researchers found a niche using socio-technical frameworks in projects to deploy new computer systems in organisations. However, Baxter and Sommerville describe these approaches as 'philosophies', rather than as structured engineering methods.

Badham et al. [8] also trace the origins of socio-technical theory emerging from analysis of worker participation in industry. They provide a definition of 'open socio-technical systems' which are

- Systems with interdependent parts.
- Open systems that adapt and pursue goals in external environments.
- Systems that possess an internal environment made up of separate but interdependent technical and social subsystems.
- Systems that have equifinality - goals can be achieved by different means.
- Systems where performance depends on joint optimisation of the technical and social subsystems

They describe socio-technical systems thinking as a framework that represents organisations as purposeful systems, and that emphasises the importance of understanding social roles and technical variance when analysing or re-designing systems. This framework is used for purposes such as work re-design, process re-engineering and computer-supported work. They also note that the general field of socio-technical theory is divided, with differences between research groups over issues such as the level of involvement of workers in socio-technical research projects and the level of autonomy implied by equifinality.

In the same volume Hendrick [74] defines a socio-technical view of systems as 'viewing organisations as open systems engaged in transforming inputs into desired outcomes'. Such a system consists of three major components - the technological system, the personnel subsystem and the relevant external environments. In this view, the technological and personnel subsystems are mutually interdependent and operate under joint causation, both being affected by events in the environment.

Herrmann and Loser [75] define socio-technical processes as 'comprising the interdependencies between persons, especially the mutually dependent activities of multiple persons', with

socio-technical systems also having ‘a technical side where artifacts, like computer systems in computer science, are relevant.’ Unusually, this definition allows for socio-technical processes that contain no technical elements.

Baxter and Sommerville [11] also note that socio-technical research exists in at least four different research communities:

1. Researchers in work, workplaces and working conditions. This includes the early origins of socio-technical systems theory, and aims to produce a more humanistic and successful workplace.
2. Researchers in information systems. Information systems are large-scale systems that support organisational and community work, and so socio-technical issues are significant.
3. Researchers in computer-supported cooperative work (CSCW). This considers the details of work and the interactions between working practices and computer systems.
4. Researchers in systems engineering. This community studies the relationship between human interactions, organisational structure and systems failure.

They also note that there is little overlap and collaborative work between these distinct communities, with research and reviews in each community showing little awareness of the others. Additionally, there is ‘considerable variation’ in the exact meaning of the term ‘socio-technical system’, with a particular divide between its origins in organisational psychology and its adoption in management science and information modelling. This contributes to the fragmented nature of socio-technical systems research; socio-technical studies draw on methodologies and concepts from a wide range of fields, and similar concepts appear under a range of different names in different areas.

The boundaries of socio-technical systems research are not well defined, especially given the existence of multiple, generally unrelated fields of socio-technical study. In particular, the term ‘socio-technical system’ is often used to describe systems that match the definitions given above, but without the philosophical implications arising from the term’s origins in participatory work studies.

Baxter and Sommerville define a discipline of ‘Socio-technical systems engineering’, which encompasses the fourth of their listed research communities. They note that failures in large, complex systems are not primarily failures of technology, but fail because they are unable to prepare systems for the social and organisation complexity of their environment - which manifests as problems such as poor requirements specification and ineffective user interactions. This can be addressed by integrating the socio-technical perceptive into systems

engineering, although it is important this occurs throughout the entire lifecycle of a project and is not just used to define requirements at the start and analyse the finished system at the end.

Baxter and Sommerville go on to identify five important research areas in socio-technical systems engineering (STSE): STSE processes, modelling and abstraction, integrated human-centric design, organisational learning and global systems. For modelling and abstraction they note the shortcomings of current modelling techniques and abstractions, which are not sufficient or intuitive enough to capture socio-technical behaviour. Additionally, they pose the general question of whether organisational modelling can actually deliver useful information for the purpose of socio-technical design. For global systems they note that STSE approaches almost universally assume the context of a single integrated organisation, whereas many systems are actually multi-organisational and geographically scattered.

Outside of specific research into socio-technical systems development itself, many research areas show either the influence of socio-technical thinking or provide methods and techniques that support the development and analysis of socio-technical systems. Requirements engineering is often concerned with the specification of complex socio-technical systems; most requirements engineering methods incorporate techniques for capturing socio-technical activities, and many case studies are set in socio-technical domains. Similarly, contemporary safety and security analysis techniques explicitly contain complex models of socio-technical interaction, acknowledging that the technical aspects of safety cannot be separated from the human and organisational factors surrounding their operation. Both these disciplines provide techniques for modelling (and abstracting) socio-technical systems; in this sense, there is considerably more research on socio-technical modelling occurring outside the field than directly within.

2.3 Early requirements engineering

Kotonya and Sommerville [102] describe requirements engineering as '[a term] to cover all of the activities involved in discovering, documenting and maintaining a set of requirements for a computer-based system'. Problems in requirements engineering can include systems that are under-specified (requirements are vague or missing) or over-specified (requirements are too detailed, restricting implementation choice or limiting flexibility); systems that include contradictory, ambiguous or unachievable requirements and systems where the achievement of requirements cannot be accurately assessed.

The difficulty of accurately constructing requirements (and successfully mapping them to designs and implementations) for software systems is not a newly identified problem, even as solutions continue to be sought. The term 'requirements engineering' was in use from

the mid-1970s [196], as large scale projects encountered problems with their requirements. For example, in 1976 Bell and Thayer [15] reported an analysis of problems occurring with software requirements. They had been commissioned by the United States government to investigate this issue in order to improve the software engineering processes of large military projects. They examined both small-scale projects (a two-month exercise set for graduate students) and a large scale project (a one-million instruction component of missile defence software). Formally submitted problem reports were studied to identify the frequency and category of requirements problems.

Both cases had what the authors considered large amounts of identified problems. Approximately one reported error per page of requirements was detected in the small project, and one error per two pages in the large project. In both cases these are likely to be underestimates, as they do not include problems addressed informally, or minor issues left unreported. Both projects also had similar types of problem - primarily requirements deemed incorrect by developers, followed by incomplete specifications and unclear requirements (although the proportion of unclear requirements tended to decrease rapidly as projects developed).

In particular, the authors note that a number of requirements problems in the missile defence system could result in a non-responsive system - opening a large gap in the strategic defence of the United States during the middle of the Cold War. Drawing from their experience, the authors then recommend additional effort on identifying requirements problems early in the development process, and note that requirements problems appear universal across a wide range of systems.

The consensus that requirements problems are a significant and pervasive cause of software failure has been established by decades of further empirical study; for example, Van Lamsweerde [187] reports on multiple large-scale industrial surveys reporting that in failed projects around the half the issues were attributed to requirements.

Early work addressing these requirements problems led to the development of SREM - the Software Requirements Engineering Methodology [5]. This approach is based around the notion of bottom-up design - that systems operate by coordinating low-level elements, while requirements are normally specified top-down by decomposing abstract functions. This, it is argued, leads to inconsistency and significant effort in transforming the requirements into a useful form.

SREM consists of two main elements - R-nets for expressing functionality, and the RSL (Requirements Specification Language) markup language for meta-requirements and analysis. R-nets are essentially state machines - diagrams consisting of transformations, path selections etc. Evaluation is performed by considering possible paths through the R-net, and this can be used to obtain values for performance and accuracy requirements.

RSL can be used for a wide range of additions to the R-nets. They can be used to describe

elements, specify transformation or establish constraints on the system. These statements are fully machine-readable, and hence enable automatic analysis. Refinement of top-level goals can be automatically checked and data flow consistency can be checked.

SREM is representative of many requirements engineering methods of the period, in that the notation employed lacks support for refining detailed specifications from higher-level objectives. Additionally, SREM's support for expressing non-functional requirements is extremely limited. The state machine structure is suitable for a number of project types, especially real-time systems; however, it is not suited to expressing the context of complex systems involving human elements.

By the 1980s, a wide range of requirements engineering methods were in use, based on a range of approaches. In a review, Roman [151] identifies six different approaches - state machines, dataflow models, stimulus-response paths, communicating concurrent processes, functional composition and data-oriented models. Recent research had also identified the need for broader coverage than earlier approaches. The need to model both the environment around the system as well the system itself had been determined, while the difficulty of expressing non-functional requirements was well known. Also noteworthy was the beginning of movement away from the 'waterfall' approach of fixed steps of development - rapid prototyping was showing some benefits, and required changes to requirements engineering approaches.

In conclusion to the review, Roman identifies the core shortcomings in requirements engineering as a lack of formalisation. Non-functional requirements still lacked a theoretical foundation to match functional requirements; higher levels of automation were clearly desirable, but this required more formality. Methods for formally capturing the full lifecycle (including user interaction and maintenance) were still lacking, and the fragmented nature of the field led to much duplication of research work.

Requirements engineering would often be integrated into full-cycle development methods, as part of an increased focus on traceability and integrated development. For example, SSADM (Structured Systems Analysis and Design Method) [44] was a UK government approved methodology introduced in the mid 1980s incorporating techniques for problem definition and requirements specification.

SSADM can be initiated with a feasibility phase, although this is optional and was only introduced in later versions of the methodology. The feasibility stage is focused on analysing the current system(s) and the operational context to identify problems or areas for improvement, rather than working to pre-specified goals. For this, two types of modelling are used - data flow diagrams and logical data structure diagrams; these modelling approaches are key to the entire SSADM technique. SSADM uses standard data flow diagrams (Figure 2.2), illustrating data flow between processes and datastores; logical data structure diagrams are more general,

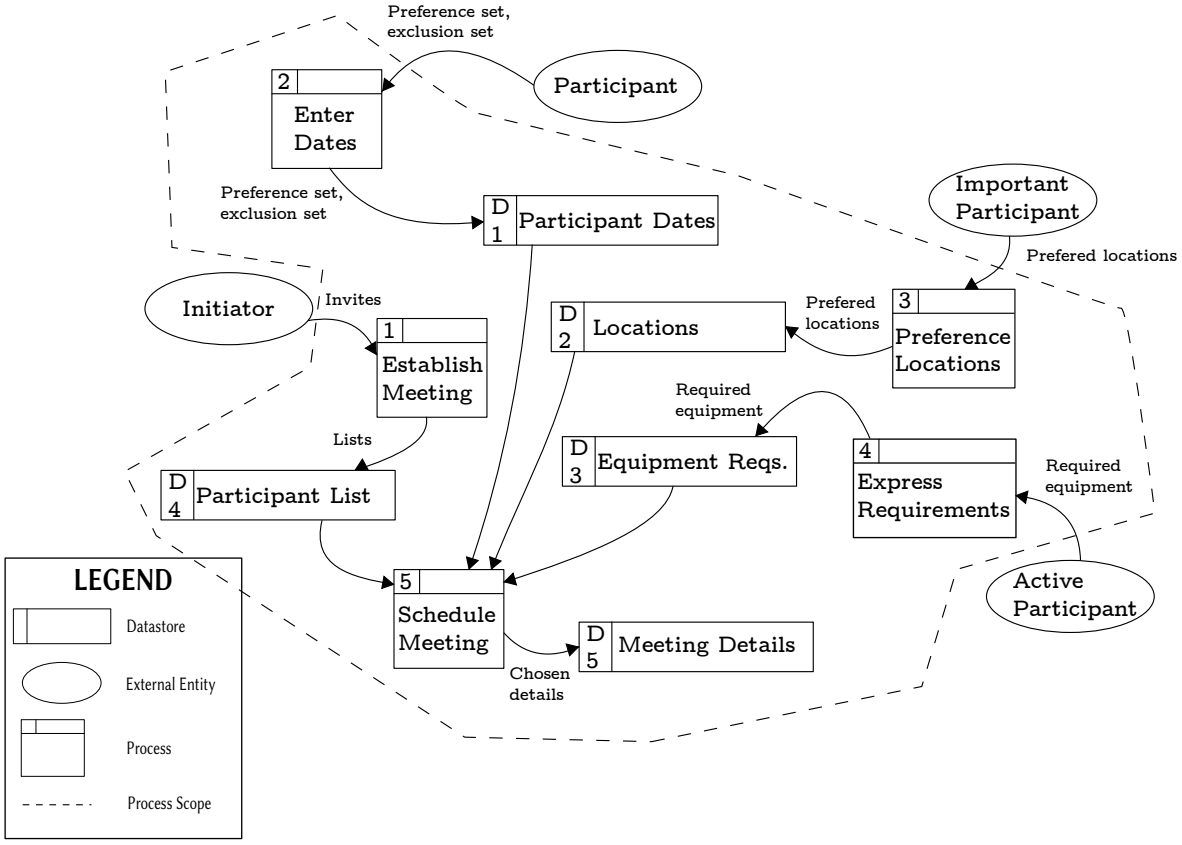


Figure 2.2: SSADM Data-flow diagram for the meeting scheduler described by van Lamsweerde et al. [188]

and are essentially entity-relationship diagrams.

The core 'specification of requirements' phase proceeds in much the same way. Initially, data flow diagrams should be constructed, based on the view of the current system. This should produce a set of required functions, as well as any potential problems. Meanwhile security and control requirements (essentially, NFRs) should be specified separately. Both these steps are then combined with more abstract requirements generated earlier to produce a draft requirements list. This is likely to produce several different potential 'system specifications'; discussion with users and the production of high-level dataflow diagrams should allow these to be reduced to a particular system option. The requirements section continues by producing highly detailed dataflow diagrams, catalogues of commonly used functions, logical data diagrams etc. This leads to the lack of a clear division between requirements and design, aiming to ensure the feasibility of requirements at an early stage.

2.4 Formal approaches

Structured requirements engineering techniques such as SSADM provided clear methodologies and notations for designing and analysing systems, but confidence in the final result was

simply based on confidence that each step of the methodology had been followed correctly. It was not possible to 'prove' the correctness of the results, either in terms of internal consistency or the relations between the specification and the implementation.

The desire for more powerful requirements specification techniques was met by the use of formal specification methods such as Z [90] [171], and VDM [95]. These approaches use formal semantics and defined rules or laws to specify allowable system states and behaviour. These approaches are more complicated and time-consuming than structured or informal techniques and so are not necessarily advantageous for systems considered to be low-risk; however, they are useful for problems of great technical complexity, and are almost universally used for (safety) critical systems [19].

The Z notation is a model-based notation for system modelling; systems are represented by defining a state, and a range of operations that can modify that state. Z is underpinned by set theory and first-order logic. While initially intended as a 'pen and paper' notation a wide range of tools have been developed to support Z, allowing much of the formal reasoning (decomposition, correctness proofs etc.) to be automated [90].

Core features in Z include set definitions and operations (for defining variables and types), standard logical operators (for conditions on state operations), relations and functions (for modelling complex data relationships and transformations) and sequences (for ordered data types). The notation is extensible, and many additional features are available in different versions of the notation. Models are constructed by defining states and transitional operations on those states. An example of the Z notation is shown in Figure 2.3.

When applied to requirements engineering, Z allows clear, unambiguous and concise statement of requirements and specifications. Requirements can then be reasoned over using formal logic so that system-wider properties can be checked. [144]. The resulting formal specifications require a good level of mathematical or logical background to understand, but the formal elements should be accompanied by natural language descriptions. Z specifications are not in themselves executable, but guidelines exist for semi-automated conversion of Z statements into high-level programming languages.

While offering many advantages when used on the right types of projects, Z and similar notations do have disadvantages. They are understandable by relatively few stakeholders, and it is possible for the formal and natural language descriptions to be contradictory [11]. The scope for Z application is limited; it is not really suitable for modelling the complexities of human behaviour, such as in social-technical systems. Modelling such systems using Z is possible, but difficult - the inherent imprecision of social actions is not easily expressed using Z concepts such as types and logical predicates.

More recently, Collinson et al. [32] propose a contemporary simulation model language called Core Gnosis, which is intended to model large-scale information systems. Core Gnosis draws

State

$invitees : \mathbb{P}\, PARTICIPANT$
$initiator : \mathbb{P}\, PARTICIPANT$
$meeting : \mathbb{P}\, DATE$
$exclusionSet : \mathbb{P}\, DATE$
$preferenceSet : \mathbb{P}\, DATE$

SetPrefs

$\Delta State$
$excludedDates? : \mathbb{P}\, DATE$
$preferencedDates? : \mathbb{P}\, DATE$
$agent? : PARTICIPANT$

$agent \in invitees$
$exclusionSet' = exclusionSet \cup excludedDates?$
$preferenceSet' = preferenceSet \cup preferencedDates?$

Figure 2.3: Z state declaration and SetPrefs model operation for the meeting scheduler described by van Lamsweerde et al. [188]

on formal process calculi such as SCRP [31]. Its core entities are processes, resources and locations; these are used to produce executable models that use stochastic elements to simulate system behaviour.

Core Gnosis is demonstrated using a boat docking example - boats arrive and must be assigned a berth and crew from a small supply. Resources can be located, such as creating 'secure' berths for certain types of boats. While nominally a real world example, this scenario is idealised such that it represents a classic resource allocation problem and the wider applicability of the technique is uncertain.

Once written, Core Gnosis models can be translated into the LSCRP process calculus (a restricted subset of Core Gnosis must be used, but Collinson et al. [32] argue that all important elements are captured). This in theory allows Core Gnosis to act as an entry point to formal modelling, providing the power of process calculi while being simpler to write and more broadly applicable. However, the core primitives remain only really suitable for systems where correct operation (according to the process specification) can be guaranteed, hence limiting its wider application.

2.5 Goal-oriented approaches

The inability of formal specification languages to model higher-level concepts inspired the development of several approaches that used goals as the main focus of their methodology. Dardenne et al. [36] first proposed a 'Goal-directed' approach to address problems with existing techniques for requirements acquisition, the initial step of developing a requirements specification. They identified several gaps in the requirement modelling tools of the time, which the goal-directed approach would address. At the time, most specification methods focused almost exclusively on functional requirements, which meant that non-functional requirements had to be recorded purely as free text, or forcing non-functional requirements to be modified in order to fit the available language constructs. Additionally, the formal specification languages were intended primarily for use by system designers, and therefore produced highly technical models. These models could not be understood by clients and stakeholders, which limited the ability to discuss and reason over design decisions. The authors created an specification language known as KAOS (Knowledge Acquisition in autOmated Specification) to address this.

KAOS modelling operates on three levels - meta, domain and instance. The meta-model describes a set of rules and standard relationships for constructing KAOS models. The domain model is the main level used for requirements elicitation and is used in the KAOS goal-directed elaboration process. Instance models represent the overall system view from the domain model instantiated with actual actors and systems. In order to obtain the full strength of the formalisation both the domain and instance models must obey the constraints set in the standard meta-model.

First-order objects in a KAOS model include entities, events and agents, alongside multiple types of goals. Entities are used to model any autonomous objects in the system domain at any level of abstraction; agents differ from entities in that they can process actions and have choice on their behaviour. These are linked by relationships such as responsibilities, wishes, inputs/outputs and conflicts. While sometimes represented graphically (Figure 2.4), KAOS models are formally represented in a specification language (Figure 2.5) that is similar to contemporary formal languages like Z.

The KAOS specification language uses concepts, relationships and attributes; these can be related as parts of a formula, which itself can be asserted using state-temporal logic (assertions may hold in past, present or future system states). At the acquisition (system modelling) level the abstract concepts of the specification language are partially instantiated, introducing new primitives such as events, actions and agents. These are instances of concepts, while relationships include inheritance and forms of composition. At this level, the details of goals and constraints are expressed using logical operators such as implication, boolean algebra and universal and existential quantifiers.

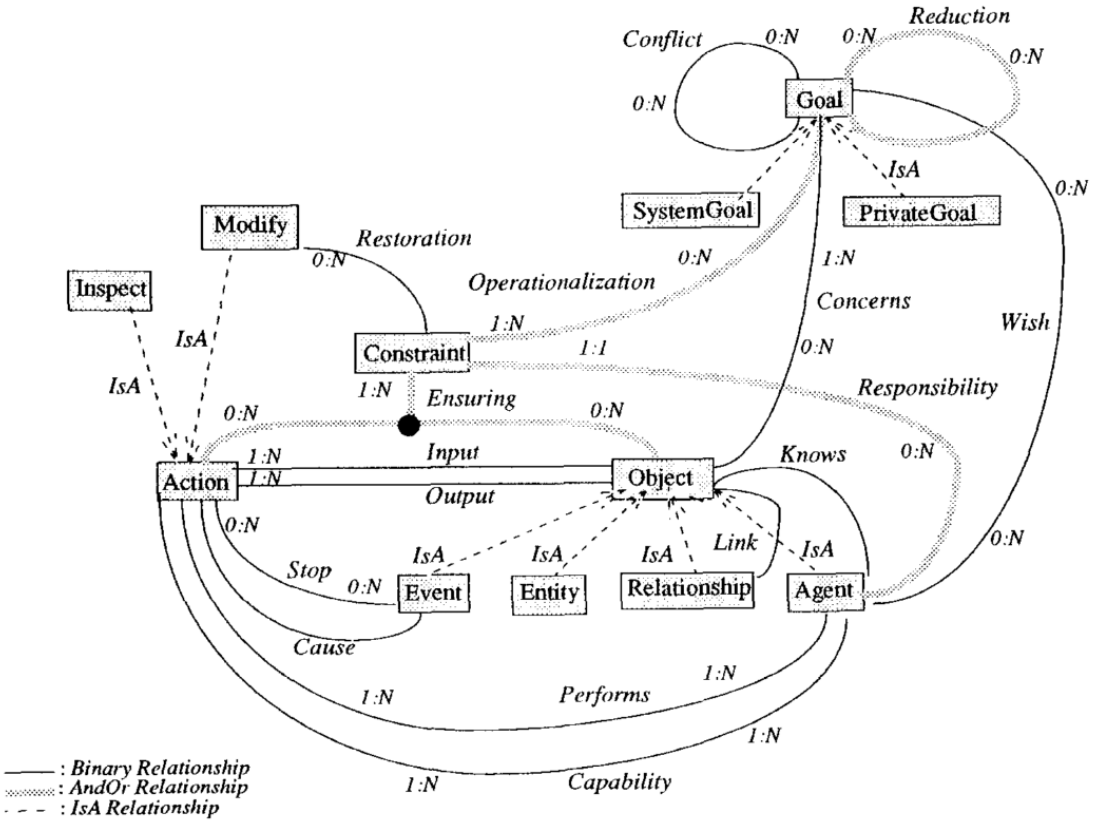


Figure 2.4: Fragment of KAOS meta-model (from [36])
Reproduced by permission of Elsevier

The graphical notation provides an overview of the model, but the majority of important details can be only be represented in the textual temporal logic. It is unclear whether graphical and logical representations of the same system need to be equivalent; case studies tend to only use the specification language [189].

The standard KAOS requirements acquisition method consists of seven steps. The first step is to identify the goal structure, by decomposing high-level system goals into more primitive goals that can later be operationalised (converted to constraints on a system state). Secondly, the potential agents in the system should be analysed to ascertain their capabilities for controlling the system. This is followed by the operationalisation of goals, where goals are converted into constraints on the system state that can be manipulated by system agents.

By this stage new objects and actions may well have been introduced into the model, especially during the operationalisation process. As a result, the fourth step is to refine and re-analyse the set of objects and actions within the system to include new objects and new actions on existing objects. The refinement carried out at the fourth stage may itself lead to model changes that mean the previously defined constraints may no longer hold, so the fifth step is to refine the constraints, strengthening objects where necessary.

This is followed by the analysis of potential responsibility assignments - the identification of

Relationship Borrowing
Links Borrower {**Role** Borrows, **Card** 0:*N*}
Book Copy {**Role** Borrowed By, **Card** 0:1 }
% Borrowers may have no copy borrowed, and may borrow several copies at same time; copy may not be borrowed, and may be borrowed at most one borrower %
Invariant ($\forall$ lib: Library, bor: Borrower, bc: BookCopy)
[Borrowing(bor,bc) $\wedge$ bc $\in$ lib $\Rightarrow$
bc $\in$ lib.checkedOut $\wedge$ $\Diamond$ Requesting (bor,bc)]
($\forall$ lib: Library, bc: Book Copy)
[bc $\in$ lib.checkedOut $\Rightarrow$
($\exists$ bor: Borrower) Borrowing (bor,bc)]
...

Figure 2.5: Sample of KAOS specification language (adapted from [36])

the range of agents that have suitable capabilities to enforce the constraints. The aim here is to identify all the possibilities and hence formalise the different design options. Finally, the model is completed by selecting the optimal assignment of constraints to agents, taking into account issues such as load and agent reliability. Once complete, the model should represent a system design that meets the original project goals at a suitable level of abstraction to begin implementation.

Tool support for KAOS was provided by the GRAIL environment [37], which provided a hybrid graphical and textual representation editor, multi-user concurrent editing and limited support for integration with other CASE tools. Models could be constructed either visually or textually, although formal assertions could only be edited in text form. The initial version of GRAIL only provided for creation and editing of models, and could not perform any analysis.

Later versions of GRAIL provided some level of automated analysis [38], although it is unclear what specific analysis methods were implemented. Notably, some functionality was added for capturing source material such as interviews, so that specified goals and entities could be traced back to their origins. GRAIL was eventually succeeded by the Objectiver commercial toolkit [150], which is primarily a cosmetic update to GRAIL without significant additional functionality.

A large number of extensions and variations to KAOS have been published, often attempting to integrate other analysis concepts or methods with core KAOS [73, 101, 131].

Nakagawa et al. [131] developed a tool that converts KAOS specifications into the VDM++ formal specification language. Goals and requirements are not directly converted, as the tool operates on the derived elements: operationalised constraints and entities. The KAOS pre- and post- conditions are used to generate corresponding VDM++ input and output specifications and hence the actual body of the function specification must be completed manually. The converted version can be used with a wide range of VDM++ tools, enabling additional forms of analysis such as verification of the specification using test cases.

Koliadis and Ghose [101] proposed a methodology called GoalBPM, which links KAOS with business process management models to improve the control and traceability of process evolution. Firstly, models of the relevant system are constructed in both the Business Process Modelling Notation (BPMN) [1] and KAOS. Relationships between the two models are then established by creating traceability and satisfaction links. Traceability links are made between BPMN activities and KAOS goals that affect that activity; satisfaction links between processes and goals indicate that the process must meet the linked goal. The trajectories of the BPMN model are then analysed, determining all possible paths through the system and their state on completion. These paths can then be traced through with links to the KAOS goals, allowing the performance of each path to be rated based on the satisfaction of goals. This allows refinements to be made to the business processes, although the process of creating a linked goal model requires significant effort.

Heaven and Finkelstein [73] produced a Unified Modelling Language (UML) profile for KAOS, providing mappings between the outer layer of KAOS and corresponding UML representations. Conversions for the KAOS temporal logic are not provided, but the authors suggest either mapping into Object Constraint Language [191] expressions or tagging objects with logic expressed in natural language. This conversion allows the use of KAOS for early-phase requirements to be integrated into the wider development process and enables low-level requirements to be traced back to goals. A demonstrative case study of a light rail train control system is used to show that the full range of KAOS constructs can be captured in UML.

In comparison to the large number of publications on tooling and extensions there are relatively few examples of KAOS case studies and evaluations in the published literature. The original authors of KAOS published a case study in which they performed requirements elaboration for a meeting scheduler [189]. This was intended as a exploratory exercise (although based on actual scenarios), and the authors combined the roles of client, domain expert and requirements analyst together. This allowed them to provide a wide-ranging view of the elaboration process, but does not reflect real-world use of these techniques. Their case study acts more as a worked example of the methodology than as a genuine application of the technique; it is a helpful guide to using KAOS, but provides limited insight into general usage. However, some weaknesses in the KAOS method are identified, including the inability to capture non-functional requirements such as usability and the need to formulate (and change if necessary) assumptions to model different versions of the system.

A study by Duboc et al. [47] described the use of KAOS to elicit the scalability requirements for a large financial transaction analysis system. This system scans large amounts of transactions and aims to detect unusual behaviour that could be fraudulent. A KAOS goal model was constructed for the system and constraints were set on performance (time to complete) and

[1] For more detail on BPMN, see Section 2.7

range (size of data input). The authors report a balance of advantages and disadvantages for this approach: they praise the traceability and responsibility relations required for KAOS, but criticise the lack of support in KAOS for ranged values and variations in assumptions over time. Generally, it is unclear if their application of KAOS provides analysis that is distinct from their domain knowledge.

Generally, KAOS fails to fully address the authors' own concerns over the inability of requirements models to be understood by wider system stakeholders, although the well-written and detailed guide to applying the technique effectively encourages standardisation and reproducibility. As such, KAOS does not offer a simpler or less formal requirements process, but does provide considerably more flexibility for capturing higher-level concepts.

2.6 Agent-oriented approaches

In the mid 1990s Yu [203] argued that existing requirements engineering approaches were failing to address the 'early phase' of requirements engineering. These early- phase requirements engineering activities include analysis of organisational goals, consideration of alternative processes and systems and addressing the interests of stakeholders - which Yu describes as the 'whys' of the requirements.

Yu argues that early-phase requirements engineering provides numerous advantages, including promoting a greater understanding of the problem domain (and hence removing a common cause of failure), addressing the increasing need for organisational integration and inter-operability and the ability to trace systems requirements back to the original organisation goals. These activities were generally performed in an informal manner, without a concrete methodology or tool support, as existing techniques were aimed primarily for later stages of requirements engineering.

To address this, the i* framework was created. The core concept behind i* is that of *intent*, which draws heavily on previous work by Yu [201]. In that paper it is argued that the common concept of treating requirements as meeting some system-wide goals is inaccurate, as individual agents within an organisation have different aims and objectives. Instead, systems can be thought of as a series of interactions between agents, who participate in order to meet their own goals. A mathematical logic for formalising these inter-agent dependencies was developed, and these relationships can be analysed to identify potential problems such as the violation of constraints.

i* analysis begins by considering the intentional relationships between actors. Actors depend on others to complete tasks, provide resources and to achieve goals. These relationships can be expressed in a 'Strategic Dependency' (SD) model, an example of which is seen in Figure 2.6. SD models portray processes as agent relationships, rather than tasks or flows of

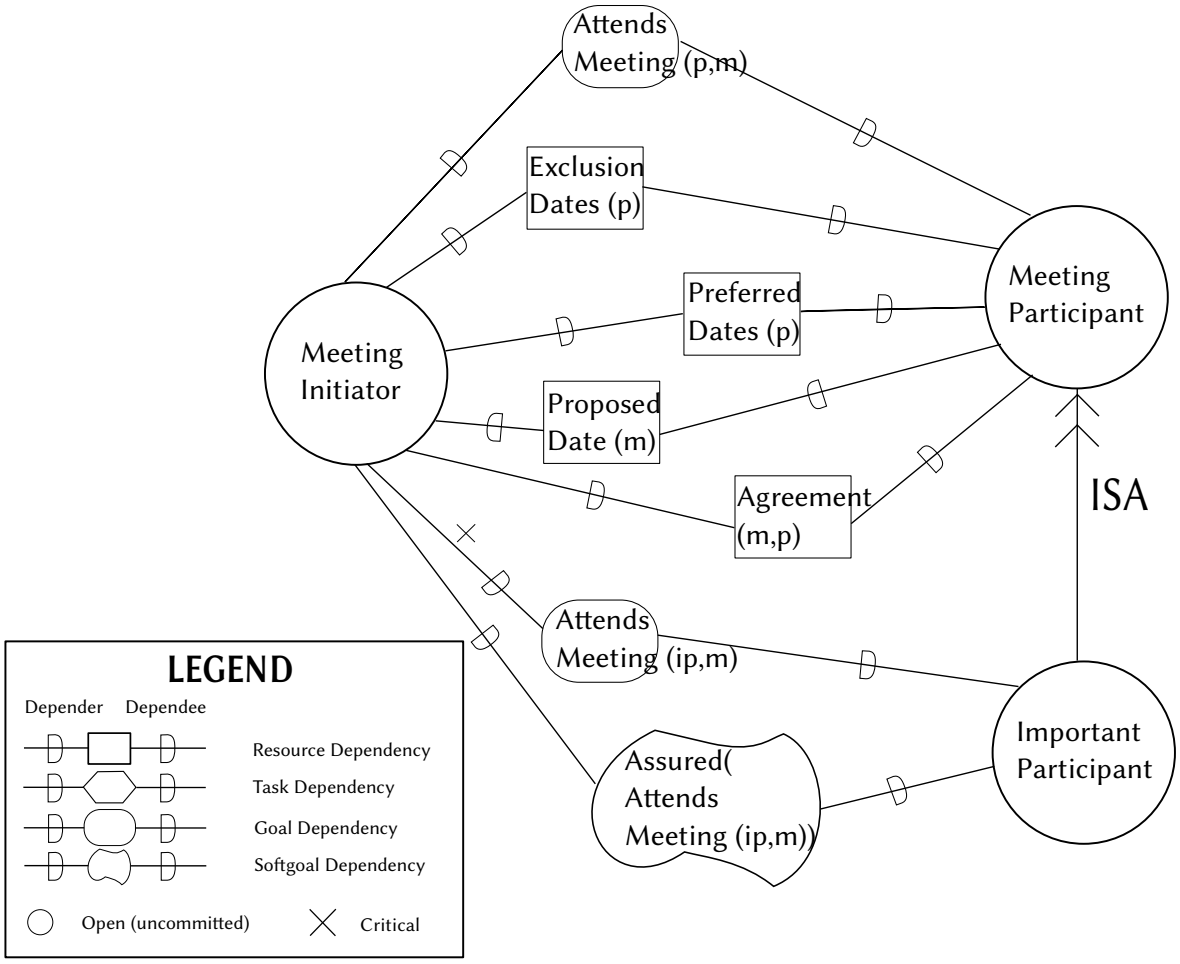


Figure 2.6: Example i* Strategic Dependency Model (redrawn from [203])

entities. This enables new forms of analysis at this level, such as opportunity (the ability to delegate/transfer goals entirely) and vulnerability (determining where actors rely on others to meet their goals).

The SD model provides a view of the external relationships between actors. Yu also argues that in early-phase requirements engineering it is often beneficial to study why actors act in the way they do and how their interests can be addressed. This can be determined using 'Strategic Rationale' (SR) models (see Figure 2.7), which use the same core entities as SD models, but decompose the internal intent of the actors. As a result, it is possible to model the internal interests of stakeholders and their attitudes towards different alternative systems.

Once routines to meet goals are specified in a SD model they can be assessed for workability and viability. The 'workability' of elements in this routine can be examined in turn and used to assess the workability of the routine as a whole. If it is not possible to accurately assess an element then that is a sign that the element should be further refined. These properties are evaluated by using a formal propagation algorithm, but the workability or viability of leaf elements must be assessed manually.

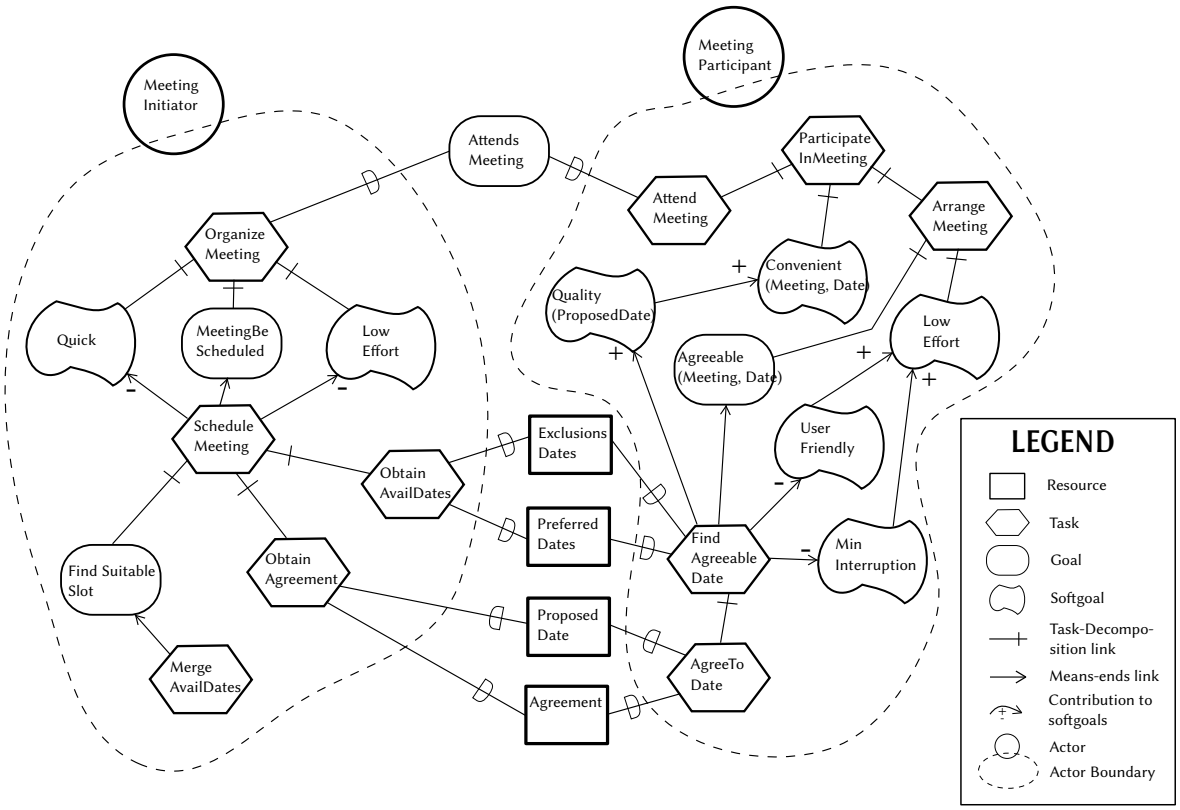


Figure 2.7: Example i* Strategic Rationale Model (redrawn from [203])

While workability assesses only the possibility of completion, viability assesses both the possibility and desirability of a routine. i* also incorporates the notion of soft goals - goals which are used to express desirable characteristics of a system rather than core functional requirements. For example, a financial transaction system should process credit card orders (a goal) and should ideally do so quickly (a softgoal). Potential routines can therefore be assessed on how many of these desirable requirements they meet and which actors benefit most from a particular design decision.

This emphasises the early-phase nature of i* - it is a technique intended to assess conflicting interests and consider high-level design alternatives, and is not a technique for modelling system designs themselves. The efficacy of i* is demonstrated in later papers, which provide a number of detailed case studies and a developed modelling methodology.

Yu and Liu [204] extended i* to address trustworthiness. Trust is considered as a special case of a non-functional requirement, and is hence modelled using a set of softgoals. The core i* set of entities is not expanded; instead, sample models are provided for standard trust relationships such as enforceable purchases and third-party assurance.

More elaborate trust relationships are demonstrated using several worked examples of a card payment system. Design variations are evaluated by considering their positive or negative contributions to softgoals. Potential attacks on the system are represented as tasks that

contribute negatively to security and trust softgoals. These relationships can be labelled with an indication of satisfaction (satisfied, denied, weakly satisfied etc.) which enables the viability of the overall system to be determined using the same propagation system as workability and viability.

As the i* core is itself not modified, this extension is more of a meta-model or extended case study than an expansion of the technique itself. The subjectivity and lack of operationalisation in assessing the trustworthiness of system components mean that this form of trust analysis is primarily suited to system scoping and early-stage design rather than modelling specific designs or existing systems.

Liu et al. [112] produced a more elaborate extension to i* by developing a framework for security and privacy requirements specification and analysis. They also propose a seven-step methodology that runs alongside the conventional i* entity identification process.

This methodology begins by identifying and analysing potential attackers, using the i* generated set of actors. All actors are initially considered possible attackers, and are considered in turn by analysing their resources and capabilities to determine potential attacks. This is followed by dependency analysis, which replaces actors in the SD model one-by-one with attackers, and traces the dependency violations to determine the reach of such an attack.

This process identifies particularly vulnerable elements of the system, so the analysis continues by identifying the different ways attackers could exploit these points. Means-ends analysis is then used to suggest countermeasures to these exploits, although it may be necessary to repeat the process to assess the security of these new countermeasures. Once complete, this methodology produces a detailed system model of potential threats and defences, which can be discussed and alternatives evaluated as with traditional i*. A refinement of the goal-labelling approach (from Yu's original paper [203]) is shown, which allows significant reduction of the solution space of possible systems by labelling links on multiple criteria and then sorting based on the relative priorities of the criteria.

Additionally, Liu et al. suggest that property verification can be carried out using the Alloy language [89] and its associated tools. They provide a partial translation of core i* entities into the Alloy language, and hence argue that rewriting i* models as Alloy specifications is practical. Once rewritten, models can be checked using the Alloy Analyser which can validate properties or produce counterexamples.

i* can also be integrated with other modelling techniques, both in software engineering and systems thinking more generally. For example, Gordijn et al. [68] propose a methodology that combines i* with the e^{3}value economic modelling technique. The combination is intended to be used to evaluate e-services: alternative business models or structures can be considered based on profitability.

The two techniques are interlocked - an i* SD model is used to produce an e^{3}value actor model, and the i* SR model is used to produce an e^{3}value activity model. The e^{3}value models are used to generate estimates of costs and revenues attributable to the various actors in the service model. The results of this analysis is then used to label the i* goals produced earlier. The labels are then propagated, and the acceptability of the model for different actors can be determined as before. If the particular structure does not ensure sufficient satisfaction for enough actors then the design process can be iterated until suitable alternatives are found.

i* has also been integrated into a full lifecycle approach, in the form of Tropos [20]. Tropos is a methodology for Agent-Oriented Programming that spans early-requirements to implementation. Conceptually, it seeks to address the same root issues as i* - the lack of attention and support for early-phase requirements engineering in software engineering processes, as well as the inability to link with later stages. Bresciani et al. [20] argue that conventional object-oriented methodologies lack documented evidence in this early-phase, citing as an example the use of UML, which begins at the use-case level.

Tropos begins with two requirements analysis phases - early and late. Both these phases are built around goal diagrams, which are essentially carbon copies of i* SR models. In early requirements analysis, they are used to describe the intentions and dependencies of actors, exactly as in i*. For late requirements analysis, the diagrams are restructured by introducing the system under development as an explicit actor. The new model is then used to assess softgoal satisfaction etc. , as per i*.

Architectural design of the system is carried out by decomposing actors and goals from the previous stages, and generating extended actor diagrams for the various subsections of the system. The relevant capabilities actors need to meet all their goals are identified, and hence a set of requirements for each sub-section of the system is produced. These can then be modelled in conventional UML approaches (activity diagrams, sequence diagrams etc.) which allows development either using specific agent-oriented platforms or conventional programming languages.

Despite the wide academic interest in i*, there are very few reported case studies in the literature. This lack of (reported) industrial application is explicitly mentioned by Maiden et al. [119], who report a series of projects using i* they carried out for Eurocontrol (the pan-European air traffic control body).

Maiden et al. developed a large-scale requirements process called RESCUE for Eurocontrol that included i* as one of four main elements - i* is used (as a follow-on from context models) to determine and elaborate system goals and rationale as well as establishing system boundaries. The RESCUE methodology was then applied in three air-traffic control projects.

In the CORA-2 project, i* Strategic Dependency models were used to determine the system requirements and to examine the system boundaries. The final SD model consisted entirely

of goal and resource dependencies and was highly centralised, with more than 90% of dependencies involving the new CORA software. These design decisions were identified as originating in the routines and mindset of the analyst that modelled them. Dependencies were adapted directly from information flows (leading to high resource-goal dependencies), while the project team were focused on the development of CORA as a software system, rather than on its integration with the wider user environment (leading to the centralised model). The use of i* helped to identify these examples of closed thinking, but the report also emphasises the high level of subjectivity and 'systems view' in the i* modelling approach.

In contrast, an i* model for the DMAN aircraft scheduling system showed that while the new system under development was involved in a majority of dependencies there were also substantial dependencies between non-system agents (Maiden et al. attribute this to a clear project focus on the social-technical aspect).

In addition to describing the industrial use of i*, a series of recommendations based on this experience are presented. These include useful applications of i* (such as heuristics for determining the system boundary), techniques for communicating i* to stakeholders (free text and tables) and suggestions for future work (such as developing superior tool support). The authors also reflect generally on the effectiveness of i* in industrial environments. They report that i* models (Strategic Rationale models in particular) are large and difficult to manage and time consuming to develop. They also highlight areas where SR modelling is particularly effective - use case discovery, soft-goal contributions and dependencies between use cases. As a result, they recommend continued but more selective use of SR models in the development process, focusing on areas that i* can model but use cases cannot.

Many i* extensions in the literature are essentially meta-models for particular problem domains, demonstrated through small case studies [112, 204]. As a result, these meta-models are rarely assessed independently, and hence their effectiveness is not easily assessed. A study by Strohmaier et al. [180] investigated the use of patterns to improve i* modelling. This study consisted of creating abstract i* models for specific types of system (such as a generic model of a collaborative editing system) and then using these models as a building block towards modelling real systems. The study found that there was no development speed advantage in using generic models, and using these patterns increased the size of models (potentially increasing model coverage, but also adding complexity).

Pastor et al. [140] carried out a similar study on core i* to assess the methodology on a number of criteria generated from previous studies of agent-oriented methodologies. Their study consisted of three small development groups (one with a modelling background, one with a programming background and one with an i* background) who each used i* to separately model three different business processes, all drawn from an industrial partner. Once the modelling was completed, the groups were asked to rate i* on given criteria and discuss their

reasoning.

i* was well rated for the wideness of its applicability in different domains, and the expressiveness of the technique, especially with regards to organisational structures. The evaluators were generally sceptical of i*'s opportunities for refinement (due to a lack of abstract primitives), repeatability (due to inconsistency in the notation in the literature), ability to handle complexity (because of the lack of decomposition and multi-level models) and traceability (because of the lack of guidelines when deriving SR models from SD models). i* was poorly rated for modularity (again due to the lack of abstraction layers), reusability (models cannot be reused in part, due to lack of modularity) and scalability (once again, due to the lack of multiple layers and modularisation). Pastor et al. hence highlight the lack of granularity and refinement as key extension points for i*, and suggest that Viewpoints (as discussed in Section 2.9) may provide a suitable solution. These criticisms of i* reflect the tendency for i* to be integrated with other techniques, such as with different varieties of business process modelling or in the Tropos development lifecycle, rather than be used by itself.

Overall, i* provides an expressive and widely applicable modelling technique that is limited by the lack of features for large-scale modelling, such as composition and modularity. The underlying concept of actor relationships provides a clear contrast with earlier requirement- or goal-based models, enabling a flexible and broadly applicable approach. However, i* lacks support for effective large-scale and multi-level modelling due to the absence of features such as entity composition. Additionally, several important areas of the methodology such as workability analysis and mappings between SR and SD diagrams are subjective or poorly described, which limits the rigour of modelling and analysis.

2.7 Business Process Modelling

Commercial and business systems and processes are commonly given as examples of social-technical systems [181] and a wide range of methodologies and approaches have been developed to suit this particular problem domain.

These approaches can be broadly divided into two categories - business modelling and business process modelling [67], which reflect two different use cases for modelling. Business modelling is concerned with profit and loss, stakeholders and the objectives of the business - models of this type include concepts such as value and contract; they essentially express the different commercial opportunities on offer.

In contrast, business process models express how different services will actually be provided and operate. They can be used to define an implementation, either for standardisation or for further analysis. They focus primarily on the internal actions within an organisation, with contact with the outside world generally limited to the request or delivery of a service. They

are often similar to workflow diagrams, describing the steps and internal interactions used to deliver a result.

In a business modelling scheme such as e^3value, business activity is defined in terms of value exchanges. Entities expose services (known here as value activities) with value ports/interfaces that define inputs and outputs in terms of their commercial value. This specifies the reciprocal nature of the services provided (benefits are provided in exchange for an expected return of value), although not any details about the relative values in the transaction. These approaches can be seen as defining a series of interfaces to a business; details of the interface are not provided, but the basic set of potential commercial transactions is expressed.

Aguilar-Savén [4] presents a comprehensive survey of approaches to business process modelling. It is highlighted that business process models may be created for a range of reasons, and that different motivations benefit from different approaches to modelling. Business processes may be modelled in order to develop software for them, or to automate the process in some other way - this requires modelling approaches that are diagrammatic (for ease of use and understanding) and unambiguous. Alternatively, process models may be created with the aim of refining or optimising the process under investigation. This benefits from models that can be executed or can be used with simulation tools to evaluate potential changes to the process. These may be more complex, and data capture to allow effective simulation is likely to be time consuming.

In their survey, Aguilar-Savén examines around ten different business process modelling techniques and categorises them based on their suitability for different purposes and the static/dynamic nature of the model. In this categorisation the approaches cluster into two main groups. Firstly, there are a range of static approaches that are primarily descriptive. This includes a number of graphical approaches such as flowcharts and activity diagrams. These approaches are easy to use and generally well understood, but may use incomplete or inconsistent notations.

The second group of approaches are more structured and have support for evaluating dynamic behaviour. These include a wide range of techniques with their origins in computing science and software engineering, such as UML [152] and petri nets [130]. Other techniques here such as GRAI (Graph with Results and Activities Interrelated) [27] instead originate from general business or the manufacturing industries. These techniques are highly suited to guiding software implementation, but they are complex and time-consuming.

The fragmented nature of this range of languages has a number of clear disadvantages, such as inconsistent levels of tool support, lack of compatibility due to different notations, lack of a common language used by both business and technology etc. As a result, a number of stakeholders jointly produced the Business Process Modelling Notation (BPMN) [195], a unified notation intended to be usable for both technology implementation and system

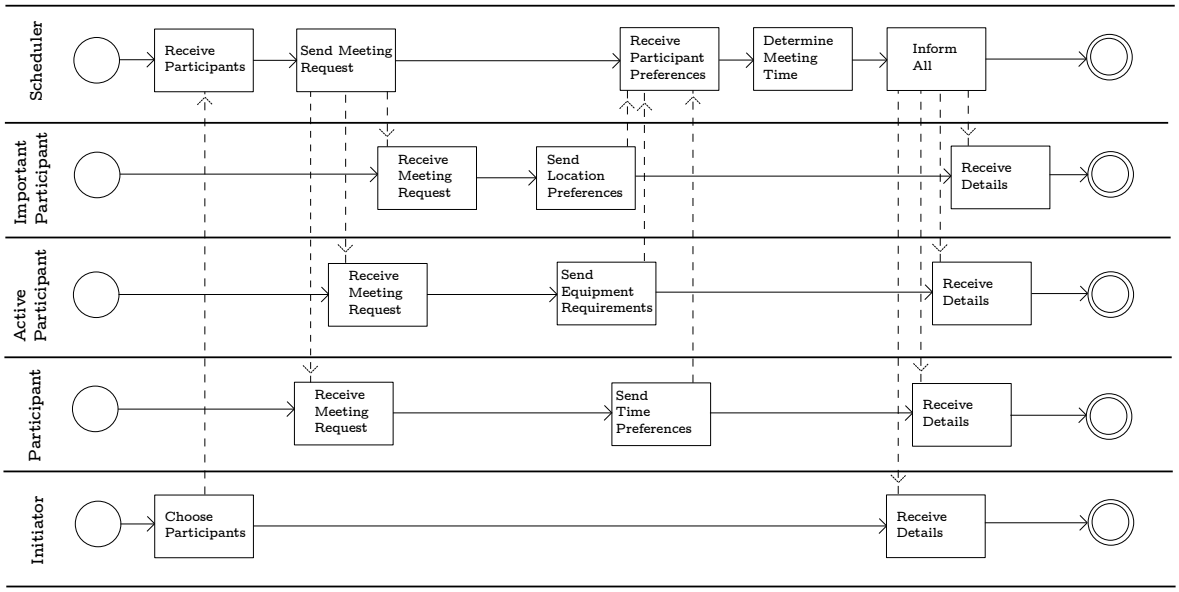


Figure 2.8: BPMN model of the meeting scheduler described by van Lamsweerde et al. [188]

analysis.

The BPMN has a small range of core elements. Flow objects are the main entities, and consist of events, activities and gateways. Events represent some occurrence during the process, and generally are either triggered by a cause or impact on some result. Activities represent any subprocess or piece of work. Gateways are used to make decisions and otherwise manipulate the flow of the process.

These objects are linked by three types of connections. Sequence flow is the standard link, and represents the order of activities within the process. Message flow is used to represent communication between separate participants (such as different business sections, or with external actors). Finally, associations offer a generic link that can be used for input/output or other miscellaneous purposes.

Organisational divisions are captured using either pools or lanes. A pool represents a distinct process participant, and communication between different pools is performed using messages. Lanes represent internal divisions within a participant, such as different divisions within a company. No messages can be used within lanes, so only sequence flow can be used. The choice of pools or lanes does retain a level of subjectivity, primarily the degree of separation between participants (consider the common use of an ‘internal market’ within large organisations). An example using pools is shown in Figure 2.8

In order to meet the needs of business analysis, BPMN models can be converted to executable form [137]. A mapping is provided for converting BPMN models into the Business Process Execution Language (BPEL4WS). The notation also allows extension through the use of artifacts, which provide additional model detail without affecting the basic structure of the diagram. These artifacts can be added based on context, such as from relevant domain-specific

libraries [195].

As with other methodologies, there are relatively few published cases studies describing the use of BPMN. Muehlen and Ho [128] report on their work with a truck repair and maintenance firm to optimise their processes and perform task reengineering. They found that BPMN models were surprisingly well understood by the firm's workers, allowing easier and quicker feedback on proposed changes. The authors also made modifications to the standard BPMN constructs to improve the expressive power and ease of understanding of BPMN models, such as allowing activities featuring collaboration to cross lanes.

Kirchberg et al. [99] demonstrate the use of BPMN to model the typical academic paper review and submission systems. They present eight different BPMN model fragments, each representing part of the wider paper review system. These show the use of BPMN for a realistic system, but do not take advantage of its capabilities for analysis or process reengineering.

In a more general study, Recker et al. [149] conducted a series of semi-structured interviews with BPMN users, testing a number of propositions regarding BPMN that they had defined based on ontological studies. Issues regarding the lack of suitable modelling constructs in BPMN (for state, history and structure) were generally not considered important by users - they tended to use BPMN for high-level modelling and stakeholder discussions and felt that the missing constructs would be more useful at lower levels. Ontological analysis also identified that the similar types of construct in BPMN (pools and lanes, or the different types of events) would lead to confusion. In contrast, users did not consider this a problem and praised the breadth of modelling options available. However, the authors note that all participants used BPMN alongside other tools, allowing weaknesses or shortcomings in BPMN to be offset by other methods.

2.8 Systems of Systems modelling

The Systems of Systems concept is defined by several authors as referring to large-scale collaborative systems where discrete sub-elements operate with some level of independence. Maier [120] defines a system of systems as a system itself consisting of systems that obey two properties: operational independence (the subsystems *can* operate independently) and managerial independence (the subsystems *are* operated independently while remaining part of the SoS). It is this second property that provides special complexity: central control is not absolute and the SoS must be prepared for variation in the performance of individual systems.

The need to model large-scale, multi-level systems and systems of systems (SoSs) has led to the development of specific techniques for systems integration such as SysML [71].

SysML [71] is an extension to the UML (specifically, UML 2.0) intended to provide tools for

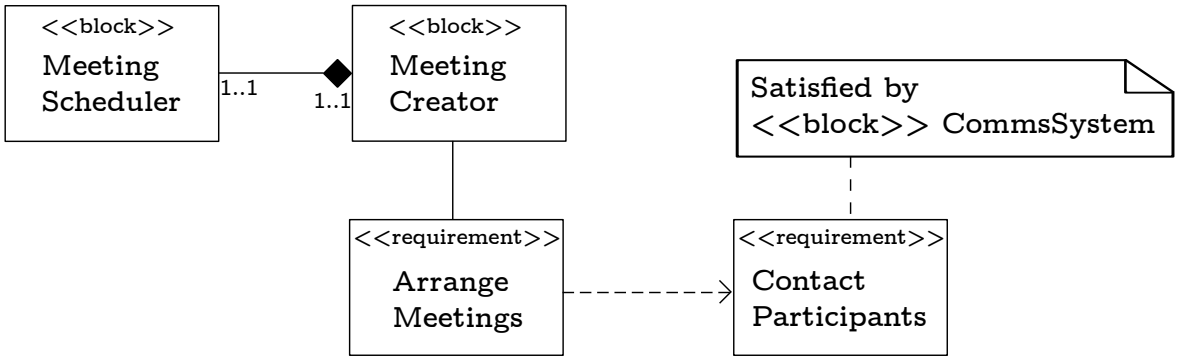


Figure 2.9: SysML Block Definition Diagram showing requirements satisfaction, composition and refinement

systems modelling as part of the wider UML framework. While core UML is very widely used it lacks several important capabilities for modelling broader systems due to its focus on 'software systems'. These include very limited support for modelling physical elements of systems and limited support for hierarchical structures. Consequentially there is limited support for explicitly modelling or validating requirements.

SysML addresses these issues by adding additional elements into UML, including new diagrams. SysML therefore consists of these additions plus a subset of re-used UML features, although in effect any UML construct can be used in SysML if required. However, the core methodology of SysML is defined as four 'pillars' of modelling - structure, behaviour, requirements and parametrics.

Requirements are specified by a completely new Requirements Diagram. A $<<$ *requirement* $>>$ stereotype is extended from $<<$ *class* $>>$ and contains a special shall statement that contains a textual description of the requirement. Relationships are defined to derive sub-requirements and to mark model elements as satisfying requirements. Rationales can also be added as tags to explain certain elements. System structure is primarily modelled by the use of $<<$ *blocks* $>>$, which are an extension of UML's $<<$ *structured* $>>$. Blocks are collections of parts (of any type) with connections that allow communication. Blocks have associated ports, which act as input/output channels and so allow systems to built up from individual blocks. At the package level the representation of blocks is black-box, possibly augmented with a parts listing or set of constraints. Block Definition Diagrams provide an internal view of blocks (which may themselves consist of sub-blocks) allowing for their creation and editing. An example is shown in Figure 2.9.

Parametrics (in the form of a Parametric Diagram) are used to define constraints on the system by the use of constraint blocks. Constraint blocks are associated with system blocks, and can define constraints on properties and values of that block. These constraints can be both environmental (such as limiting the range of inputs or defining physical laws) and internal (such as restricting certain output values from a block).

For system behaviour, standard UML approaches such as use cases and sequence diagrams are re-used. Activity diagrams are also used, but have been extended from UML with additional features for probabilistic techniques and improved decomposition.

In an evaluation, Herzog and Pandikow [76] analyse SysML to identify strengths and weaknesses. Overall they welcome the development of an systems engineering based extension to UML, and conclude that no major issues have been raised against SysML. However, they do note several minor concerns about the language. The flexibility of UML means that SysML models can vary widely in the level of abstraction and the concepts used to model systems - it is possible for critical parts of a system to be effectively hidden in the specification of a low-level element.

Additionally, they emphasise the importance of effective tool support and the establishment of a wider infrastructure around SysML; at the time of their evaluation (soon after the launch of the notation) this was limited.

Several researchers have expanded SysML to produce executable or analysable models from standard SysML diagrams. Huang et al. [81] aim to create simulation models directly from SysML using standard OMG (Object Management Group) tools and standards. Their process begins as normal by creating a domain model of the system in SysML using a block definition diagram. They then build another block definition diagram containing an abstract model of the simulation system - blocks such as queues, entities and processes are defined here. The blocks in the simulation model are linked back to the domain model using the SysML relationships generalisation and association.

Once these two corresponding models are created they are exported in the standard XMI file representation. This file is then further processed using XML tools to produce a format suitable for import into a system modelling package (Huang et al. use eM-Plant). It is unclear how much of this process requires manual intervention, and how much could be automated; additionally, it is unclear whether the simulation meta-model is suitably generic, or if it must also be modified depending on the choice of simulation software target.

Jarraya et al. [91] demonstrate a conversion of SysML activity diagrams into Markov chains that can be analysed using PRISM model checker [77]; they also extend activity diagrams to include time constraints. Firstly, the relevant system elements must be modelled in a SysML activity diagram. The two main areas that can be analysed using PRISM are probabilities and timing, and so they should be expressed in the diagram - probabilities using the standard SysML notation of labelling edges for decision nodes; timing is expressed using a simple extension of SysML that allows activities to be annotated with times that are either fixed (x time units) or equiprobable intervals ([a,b] implies the action will take at least a time units, at most b time units).

The transformation of the SysML model is performed using an algorithm that effectively

converts threads within the activity diagram into reactive modules that can be interpreted by PRISM; an algorithm is provided for this, but implementation details are not specified. Once the PRISM-readable version is produced, properties of the system can be checked by specifying them in the PTCL logic and testing them with the model checker. The probabilities of reaching specific states in the system can be assessed, as can the probability of completing activities within a specified time.

2.9 ‘Soft’ approaches

Systems engineering is a well-established discipline that aims to analyse, control and manage complex engineering systems. Traditionally, systems engineering is concerned with a relatively narrow definition of ‘system’ - typically a large process system (e.g. a chemical plant or steelworks) or an engineering development project (ranging from a space program to the development of a new railway carriage) [26]. Understandably, techniques intended to manage these types of systems were often not suitable for systems exhibiting substantial social-technical aspects.

This inspired the development of Soft Systems Methodology (SSM) [26], intended to provide a ‘soft’ form of systems engineering. The developers of SSM noted that classical systems engineering was designed to manage systems that that had well defined requirements; when the outputs were clear a system could be constructed to deliver them. However, in systems more generally this may not be the case - requirements may be more abstract, or system stakeholders may feel improvements are needed, but be unsure exactly what. SSM aims to provide a system for reasoning about such systems, enabling definition and analysis. In this regard, it shows many similarities to early-phase requirements engineering, although its application is more general.

SSM retains a high level of flexibility (and variance) as an analysis tool; it is a system for analysis, rather than a process per se. However, it has eventually converged on a set of analysis methods and modelling techniques that can be considered standard.

Initially, the system(s) to be analysed must be defined. This is likely to include both overarching systems such as ‘the organisation as a system’, as well as subsystems of relevant interest. Once chosen, these systems should be specified in terms of a root definition. The elements of the definition can be specified using the CATWOE acronym - customers, actors, transformation process, Weltanschauung (worldview), owner and environment. Fundamentally, the root definition should express the systems as a process set in a worldview that provides meaning.

Once defined, the systems need to be modelled in a series of steps. These are focused around the transformation process; setting the specific task, obtaining resources and assessing the output. This is intended as an iterative process, with several models produced, often at

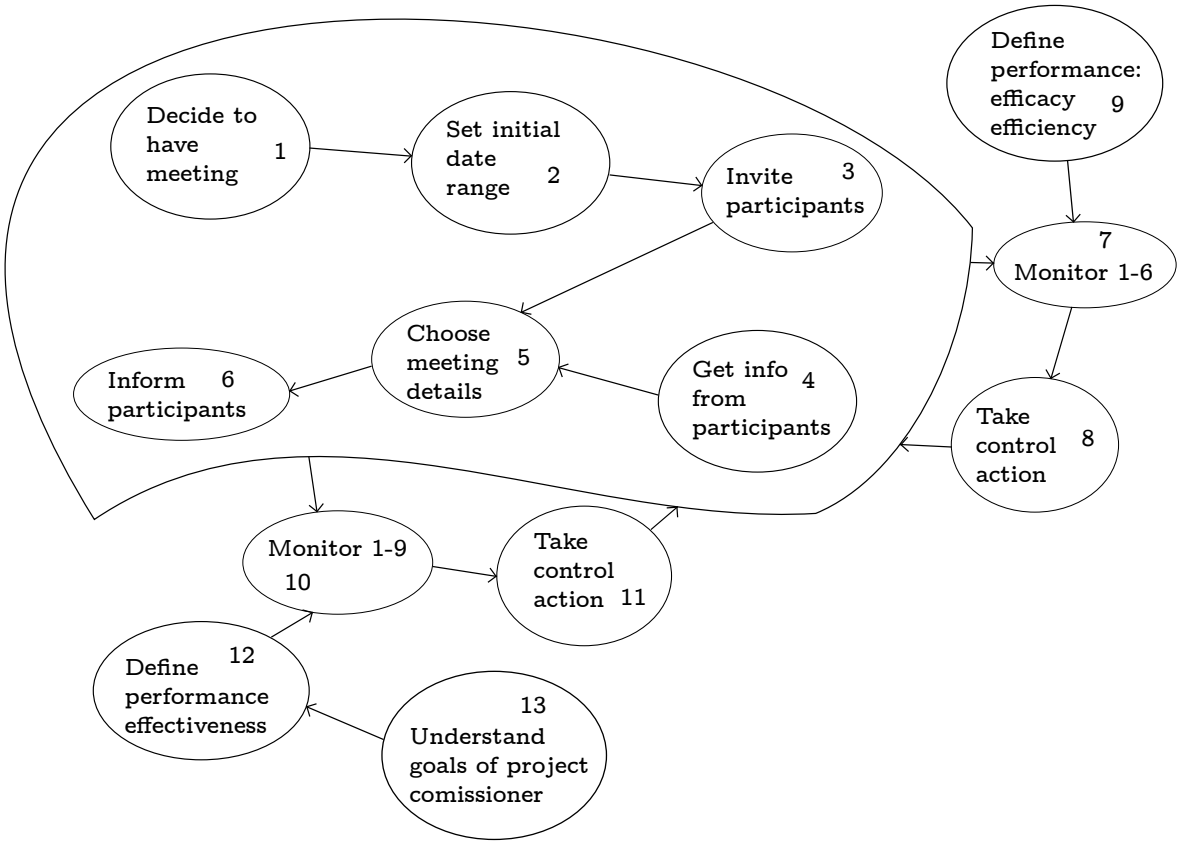


Figure 2.10: SSM System model of the meeting scheduler described by van Lamsweerde et al. [188]

different layers of abstraction. For example, additional constraints could be added, or it may be necessary to monitor performance. Classically, a two-level performance monitoring system is used based on the 3 'E's : efficacy, efficiency and effectiveness (as shown in Figure 2.10). The end result of this modelling process is essentially a (sub)task diagram, showing the necessary steps without explicitly specifying implementation details.

SSM commonly uses three standard analysis types - named Analyses 1, 2 and 3. Analysis 1 is an analysis of system roles both within the system under analysis and the roles of those conducting the study. This analysis should identify clients (who are commissioning the analysis), problem-solvers (who wish to change the system) and problem owners (system stakeholders).

Analysis 2 is a study of social elements within the system. A model of roles, social norms and values is used; techniques for this are not specified, but the aim is to consider the social interactions (influenced by norms and values) within the system.

Finally, Analysis 3 covers political issues; more specifically, how power is distributed within the system. Power may be formal (through management structures etc.) or informal (reputational, or via a network of contacts etc.). Analysing the power structures and relationships will give a clearer view of the system, although again no set methodology is provided.

SSM is often used as an initial step for problem definition and exploration before using ‘harder’ methods for later steps [142]. Design and analysis methods often assume the existence of a well-defined problem and an agreed solution, which can be obtained by the use of SSM.

The flexibility of SSM provides wide applicability, but at the cost of consistency and rigour. As a explorative modelling tool SSM is effective at identifying the core elements of a system, including important social concepts that would not be captured by conventional systems modelling approaches. Additionally, the generic nature of SSM means that it can be applied to almost any system, regardless of the underlying paradigm. However, the ‘standard’ analysis methods are poorly defined and highly subjective; they identify potentially important system issues, but do not provide a structured approach for investigating them.

Finkelstein et al. [59] propose the use of Viewpoints as a technique to unify different approaches in systems development. In systems development, different participants may use notably different views of the system during their activities - managers may see the system in terms of personnel organisation, planners may use state machines and developers use block diagrams or petri nets. This breadth of representations cannot easily be captured within one modelling notation or schema, but it is often desirable to be able to consider the system as a whole.

Viewpoints act as a framework that can be used to collect together different system views. A viewpoint consists primarily of a work plan and a specification, written in some representation style. For example, a developer may have a viewpoint represented as block diagrams; the specification consists of the block diagrams for the relevant parts of the system, and the work plan is the developer’s role in implementing and integrating their part of the system.

Viewpoints can be linked where it is necessary to check the consistency of the system. This is clearly easiest when viewpoints share a representation style, such as when developing sub-system components. However, the viewpoint structure also allows links between viewpoints of different styles - in this case, it can be asserted that two viewpoints need to be consistent, but the actual process of checking requires careful manual comparison.

Viewpoints also lend themselves to meta-construction - a system development approach can be specified using viewpoints, with new viewpoints created dynamically to represent development activities. When this approach is used a wide range of consistency checks are generated, offering a clearer development strategy.

The viewpoint approach is not by itself a development approach or modelling language; it is a framework for structuring processes in a way that encourages modularity and consistency. Complex system elements can be encapsulated away as specific viewpoints, while remaining linked to and defined by a higher-level system view. In particular, collaboration is encouraged by this approach, as sub-system links are easily defined without requiring knowledge of the implementation details, and checks can be delegated to specialists that are familiar with both

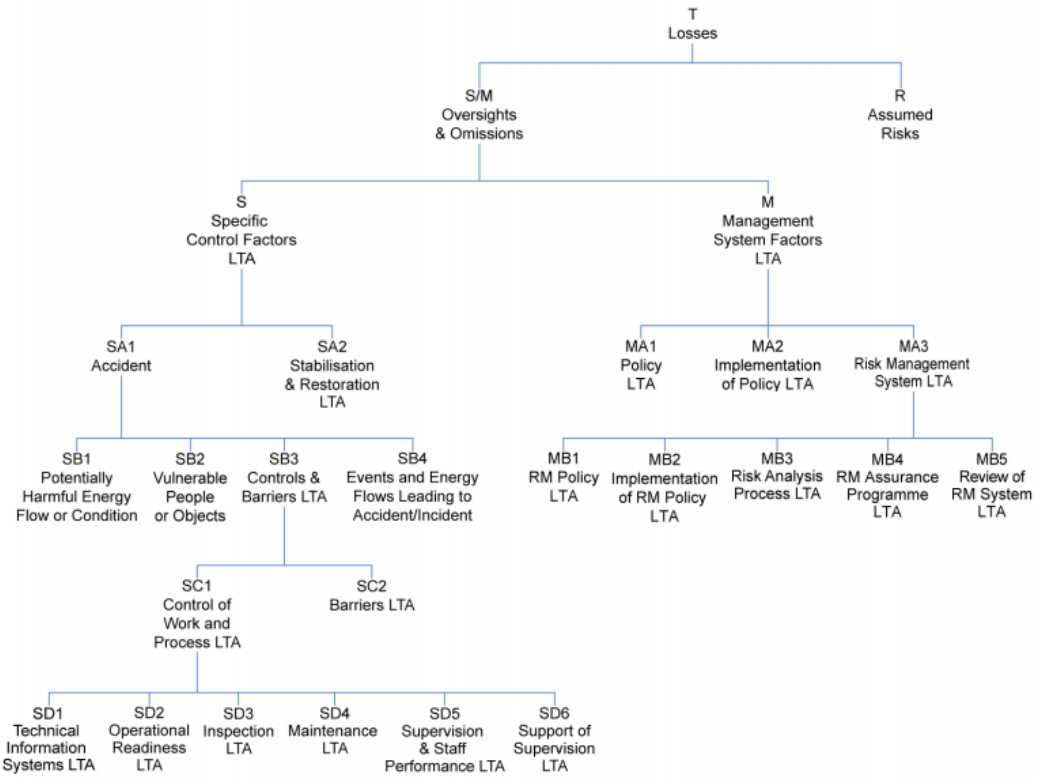


Figure 2.11: Simplified MORT logic tree (abstracted from [94])
Reproduced by permission of The Noordwijk Risk Initiative Foundation

notations.

2.10 Safety Analysis

Systems analysis draws on a wide range of techniques and concepts originally developed for use in safety analysis and accident investigation. In this context, methodologies focus on analysing failures - either to investigate after an accident has occurred, or to prevent them from occurring in the first place. Common techniques in safety analysis include cause-and-effect based techniques such as MORT [94], FMEA [23], and HAZOPS [182] as well as systems-based approaches such as STAMP [109] and FRAM [80].

The MORT (Management Oversight and Risk Tree) [94] methodology is a logic tree approach for accident analysis, drawing on lessons learnt from the use of fault trees [190]. The core of MORT is a tree model (see Figure 2.11) , which models a wide range of technical, organisational and mechanical elements of a generic industrial process. This graph of elements is used in conjunction with a set of standard questions to analyse potential contributory factors to an accident. By working through the various branches of the tree each relevant part of the wider system is examined, and sections that are ‘less than adequate’ can be identified.

The analysis itself focuses on the concept of an accident as an unintended energy transfer. While this extra energy may originate in some specific system sub-element, MORT analysis aims to determine how and if such extra energy will be effectively dissipated or redirected. Unlike many earlier accident approaches, both process elements and structural/managerial elements must be considered as a contributory factors if an accident occurs.

The MORT approach employs standard analysis questions and a meta-model to reduce the potential for subjectivity. However, other analysts such as Johnson [93] argue that MORT is highly subjective, as particular scenario conditions can be attributed to different branches of the fault tree depending on the arbitrary choices of the analyst.

The scale and breadth of coverage of the MORT meta-model potentially comes at the cost of the analysis being more time consuming; Johnson [94] suggests that simple events can be analysed very quickly, although this may require subjectivity in choosing which parts of the model to focus on. Additionally, the core meta-model is designed to address a traditional industrial process, and so modification to the approach would be required to utilise it for information- or organisation-based scenarios.

An early systematic approach to systems analysis is the FMEA (Failure Modes and Effect Analysis) methodology, which is used to analyse system safety by evaluating the potential ways system components can fail [23].

Most applications of FMEA are applied at one level of the system (usually the lowest / most detailed). All components of the system at this level are identified, and the potential failure modes of each item is recorded. Failure modes are broad classifications of unintended item behaviour (e.g. a water pump may fail entirely, reduce in output, start at the wrong time etc.), and each failure mode may be caused by several different failure causes (specific faults, such as worn-out valve, microprocessor failure etc.).

Once the failure modes are identified, the consequences of this type of failure are estimated, both on directly related elements and on the wider system. This results in the production of a comprehensive document describing the likely failures of the system, and can be used for risk assessment or to consider countermeasures for common failings. FMEA is an efficient approach for analysis when component failures have clear consequences on system performance; it is generally not suited to complex systems that involve collaborating subsystems, or where the system must be analysed at multiple layers of abstraction, as the quantity of detailed information required becomes unmanageable.

The FMECA (Failure Modes, Effects and Critically Analysis) extends FMEA by also considering the severity and likely frequency of each failure mode. The combination of these two factors allows the creation of a ‘criticality’ score reflecting the potential impact of the failure, allowing the identification of the most serious system vulnerabilities for remedial action.

Safety cases [16] are a widely used method [17] for presenting evidence of system safety. In a safety case, a claim is made about some system, asserting the correctness of some attribute. The case is structured so that an argument is presented to validate the claim; the argument is created by providing appropriate evidence, which is linked back to the claim using inference rules. Arguments can constructed in a wide range of ways. Deterministic arguments include exhaustive testing or mathematical proofs (e.g. railway signalling). Alternatively, arguments can be probabilistic, based on statistical analysis and testing runs (e.g. mean time to component failures). More abstract claims may require a qualitative argument, aiming to show that procedures or processes meet the claim (e.g. ISO9000 compliance).

The HAZOP (Hazard and Operability Study) [182] is a long-established method for safety analysis of processes. The technique was originally developed by ICI in the 1960s and 1970s to analyse chemical plants but is now widely used in a large variety of fields.

HAZOP studies examine a system by considering the processes and subsystems within it. This necessarily requires the system to be specified in sufficient detail for subsystems to be revealed, but HAZOPs can be applied to both existing and proposed systems. Once the subelements have been identified a set of guide words is used to identify potential deviations from normal operation; for example an increase in temperature or late arrival of information. The set of guide words often varies depending on context, consisting of both general and domain-specific examples.

The analysis is carried out by examining each combination of process element and guideword in turn. Many elements will offer no risk or operate safely up to very high tolerances, and these can be marked as 'no further action'. Elements that have only limited safety tolerances or offer no protection against certain deviation should be referred for further analysis and are generally prioritised by the likelihood of said deviation.

Applying this guideword approach in full on every subsystem is generally impractical, due to the exponential number of guideword-process item combinations. HAZOP guides emphasise the importance of using a skilled team to perform the analysis who can significantly reduce the time taken by disregarding unnecessary combinations.

HAZOP inspired guidewords have been widely adopted into other methodologies. For example, they have been used as part of 'risk clauses' in responsibility modelling [115], hazards in Systems of Systems modelling [114] and trust analysis in e-government systems [6].

Leveson [109] created the STAMP (Systems Theoretic Accident Model and Processes) modelling technique, which differs from other accident analysis approaches by using a systems-theoretic approach to analysis. This approach does not identify single points of failure or accident causes, but considers the interactions of the combined system elements.

STAMP is built around three core conceptual areas: constraints, emergence and process. The fundamental concept within the STAMP model is a constraint. Constraints are restrictions placed upon layers of the system by layers above them in an attempt to control their behaviour. Leveson argues that constraints can model all system levels, even at the component level. Mechanical failures can be seen as failures to constrain the manufacturing or maintenance processes.

The emergence and hierarchy concept embodies the view that safety cannot be assessed by looking at individual layers of the system - safety is an emergent property of the system as a whole. In this view, safety can only be obtained by some form of control structure that crosses system levels, and this control process is a key area of analysis for STAMP.

The third core STAMP concept is the process logic or 'process under control' model. This is a general model of process operation which can be instantiated for different systems. The model consists of the controlled process (the operative part of the system or sub-system), the controller(s) (which may be human agents or automated systems) and the actuators and sensors used by the controllers to measure and manipulate the process. This model allows the structure of process control to elaborated, and can be used to identify a wide range of common vulnerabilities.

Leveson demonstrates the technique by a detailed analysis of a failed Milstar satellite launch. The analysis incorporates the core STAMP concepts by dividing the overall launch into subsystems (Launch Operations, Software Development etc.) and identifying constraints that were violated and associated process model mistakes or control flaws. However, no defined methodology is given in the paper and there is no description of how to identify violated constraints or control flaws. The case study demonstrates that the approach can produce a comprehensive result when applied by an expert in both the STAMP approach and the subject matter, but it is not at all clear that STAMP could be effectively applied by other analysts using the published details as a guide. Additionally, STAMP appears to act more as a method for structuring information (in constraint-control-process tuples) rather than a direct technique for accident investigation or system analysis that can elicit new insights.

Accident analysis methods typically draw cause-effect relations, attempting to trace back steps to identify the origins of a failure. In contrast, the FRAM (Functional Resonance Analysis Method) technique by Hollnagel [80] aims to address situations where systems may be intractable and depend on complex linked processes (the 'second cybernetics').

The FRAM approach is underpinned by four principles. The equivalence of failures and successes states that both correct and incorrect outcomes arise from the same starting point, so it is not possible to identify obviously 'incorrect' actions without hindsight. The concept of approximate adjustments highlights the great variability of human performance, and argues that is not a negative but a positive, as it allows under-specified processes to operate correctly.

Emergence represents situations where cause-effect relationships cannot be determined due to a lack of information or excessive complexity; resonance refers to potential amplifications of multiple small performance deviations.

The FRAM method seeks to identify what is required for activities to 'go right' rather than what caused them to fail. The method starts by identifying the functions within a system (here functions refer to activities performed in an attempt to meet some goal). Functions are defined in terms of inputs/outputs, resource consumption, control, timing and preconditions. FRAM functions sit individually and are not explicitly interlinked in the style of most task-based models; functions are analysed individually and only linked when the model is instantiated.

Once identified, functions should be assessed for their variability (variance in their outputs). This can be approximated by qualitatively considering the timeliness and precision of the function's output. Alternatively, a more elaborate analysis can use HAZOPS-style failure modes on properties such as duration, sequence or speed. Once the inherent variability of function outputs is determined the next step is to evaluate the effects of variable inputs on the variability of function outputs.

This again adopts a HAZOPS-like approach, with function inputs potentially being too early, too late, imprecise etc. Most inputs are affected by variability in the same way, with late, early or imprecise inputs increasing variability, and on-time and precise inputs reducing variability. However, there are some exceptions such as the early arrival of resources having no effect on variability of the output. In contrast to some other analysis techniques, there is no separation of tasks and actors in FRAM - each function in the system has the actors involved explicitly included in the function description.

Once this analysis is complete, measures can be taken to reduce the variability of the system, and traditional techniques such as hazard prevention and protection can be applied. However, FRAM also offers a new resolution technique - dampening of variability. When much of the performance variability is caused by interactions (functional resonance) between different elements of the system, the introduction of additional checks and safeguards can reduce the chance of performance variations propagating throughout the system, providing more consistent performance.

Hollnagel [80] provides three worked examples, covering healthcare, finance and transportation. As expected, these examples closely follow the methodology as previously outlined. Functions are constructed by identifying a key point in the process of the system, and then specifying the functions needed to produce the inputs and consume the outputs of that point, continuing until the functions become trivial or out of scope.

The instantiation of the model involves producing a diagram showing the functions 'wired' together with corresponding inputs, outputs and other signals. This does not provide any formal underpinning to the system model, but instead serves as another way of visualising

the intricacies of the system and hence improving understanding. Analysis of potential vulnerabilities in the system comes primarily from the details of the function specifications rather than from the instantiated connections.

FRAM was explicitly intended to not rely on some form of internal world model, as such models within methodologies are argued to constrain thinking [117]. The system specific model created by performing FRAM on a scenario is not quantified; interrelations between functions are recorded but their magnitudes are not. As such is not an approach that lends itself to automated analysis or definite statements on cause and effect. Instead it acts as a way of structuring information on a system while still requiring domain knowledge to produce useful analysis.

Overall, the investigation of safety analysis techniques shows the difficulty of balancing modelling and analysis detail with ease of use and time-effectiveness. The advantages of structured analysis techniques such as the standard questions of MORT and HAZOPs, and the meta-models of STAMP and MORT are clear - they reduce the subjectivity of the analysis process and provide a guide for the inexperienced analyst to apply. However, the structured approach demonstrated in these techniques has significant disadvantages - the structure may limit the wider applicability of a technique (such as the underlying process model in MORT) or require an intractably large amount of time to exhaustively apply (such as HAZOP keywords). In contrast, concepts such as systems theory and functional resonance provide a useful way to reason about a system, but have inherent subjectivity that makes consistent analysis more difficult.

2.11 Deontic logic

Deontic logic is the logic of obligations and permission, and is considered by Meyer et al. [125] as a formal underpinning for system modelling. Deontic logic has operators for permission, obligation and prohibition, making it potentially useful for modelling a wide range of social-technical agent relationships. Early computational applications of deontic logic included its use for formal legal models as well as for system specification, where it was used to separate the permission (or prohibition) to perform an action from the actual effects of the action, allowing better modelling of fault-tolerant systems [124]. However, most forms of deontic logic contain paradoxes (which are internally consistent, but paradoxical when applied to real-world examples), so either special care must be taken when translating real specifications or a custom variant of the logic should be used [125]. For example, Chisholm's Paradox highlights the formal inconsistency between "ought-to-be" and "ought-to-do" statements. In standard deontic logic it is impossible to assert both that an understandable event should not happen, that the response to this event should vary based on how undesirable it is, and that

the undesirable event does actually happen. This set of occurrences is perfectly plausible (consider any emergency response system) but impossible to consistently formulate.

Cholvy et al. [29] used deontic logic to specify different forms of responsibility (see Section 2.13.1 for a discussion of the forms), where Standard Deontic Logic (SDL) is used as a natural formalisation for the duties agents hold towards each other.

Letier and Heaven [106] use deontic logic to extend labelled transition systems (LTSs) to refine the model events - enabling the distinction between obligation and permission lacking in standard LTSs. LTSs are system models consisting of concurrent components, each of which is an event-driven state machine. With conventional LTSs, events are 'forced' to occur by 'maximal process' assumptions that emphasise the machine over the environment; this assumption fails to hold in most requirements modelling scenarios, where the machine/environment boundary is indistinct and global assumptions impede modularity of design.

To address this, LTSs are extended to include transient states - states which must be left if at all possible. This enables 'forced' events to be modelled as transitions from a transient state in a way that allows modular decomposition. It is then possible to formally check if agents are capable of achieving their goals. This process involves deriving a satisfying interface and then back-propagating from error states to determine if they can be avoided by the actions of the agent. When combined by previous work by the same authors [107] this allows easier analysis of models derived from higher-level approaches such as KAOS.

Padmanabhan et al. [138] propose the use of deontic logic to model business processes; in particular, they wish to address the contractual relationships that exist between partners. They construct a system of logic using actions ('bring about'), directed obligations and proclamations as primitives and augmented with standard axioms. Proclamations are in effect public, generalised obligations and can be used to capture multi-agent commitments without the need for a dedicated primitive.

Additionally, Padmanabhan et al. propose extending i* by introducing a deontic dependency (at the SD level); it appears that this is intended to capture obliged behaviour, as the description is unclear if other deontic operators (permitted / forbidden) can be used instead.

Deontic logic conceptually appeals as an ideal formal underpinning for social-technical modelling; the concept of modelling systems in terms of actions between agents is an intuitive one. In particular, this concept fits very well with agent-based modelling. Despite this, most applications of deontic logic are focused on purely mathematical extensions or on small, state-machine like systems; Padmanabhan et al. [138] works on combining deontic logic with agent-based modelling but this provides no advantages for the modeller as areas such as logic-based model checking are not examined.

2.12 Timebands

Burns and Baxter [24] aimed to address the insufficient treatment of time in system modelling; time is normally treated as a 'flat' concept that does not provide layers of abstraction. As a result, temporal elements are often either underspecified (consider any action that must be performed 'immediately') or specified to unnecessary detail (the distinction between a 0.01s and 0.02s reaction time is superfluous in most human interactions).

To address this, they propose the idea of 'timebands' - bands that represent a certain level of time. Bands are defined by their granularity (the scale of a meaningful unit of time within the band) and their precision (in terms of distinguishing events within the band). For example, an airline schedule could be modelled using a band with granularity fifteen minutes, and precision one minute (as relevant events such as take-off and landing take ~15 minutes, but delays are counted to the minute).

Two types of occurrences may occur within a timeband. Events are occurrences that occur in negligible time with respect to the granularity of the timeband - they are effectively instantaneous within the context of the timeband. Activities are occurrences that have a duration that can be expressed in terms of the granularity of the timeband. Within a band it is possible to set precedence relations - constraints that indicate that one event or activity must occur before another can start.

Multiple timebands can exist, and occurrences can be mapped between different bands. Occurrences can be represented as events at more abstract timebands, and then by activities in the more granular bands. Likewise, activities may become events as timebands become broader. For example, the take-off of an aircraft is clearly an activity in a timeband for modelling that particular flight, but would be treated as an event in a timeband modelling the weekly performance of the airport.

This initial version of timebands provides a useful technique for modelling the temporal behaviour of systems, especially across systems that involve order of magnitude differences in temporal resolution. Separation of concerns becomes possible (both human interactions and transistor-level performance can be modelled using the same methodology) while elements at different bands can still be related to each other. It is unclear if this approach can support formal techniques such as verification of inter-band relationships, but the core concept is intuitive and addresses a genuine shortcoming in many other modelling approaches.

Having outlined the concept of timebands effort moved on to producing a working implementation. Baxter et al. [13] report on using timebands to model a hospital intensive care unit; their model was implemented using constraint logic programming.

Initially, they began by modelling a single timeband, which featured a series of alarms in the care unit. Upon executing the model the time taken to address these alarms was considerably

less than appeared realistic. This was traced back to vagaries in the specification of the model - activities were not explicitly ordered, and so the model was able to obtain a significant speedup by (impossible in practice) parallelisation of activities.

Upon implementing a second timeband to run in parallel, the time taken to resolve the incidents was again faster than expected. This time the problem originated in the actor model of the system - actors were specified independently in each band, so it was entirely possible for one (real) actor to be performing two actions simultaneously, once in each timeband. This problem appeared difficult to resolve, as the authors did not attempt to eliminate the duplication of actors, but instead ran each band separately and combined the results.

At this stage some of the classic modeling problems become apparent. Useful analysis techniques appear to be in reach - in this case, the ability to determine the time needed to perform activities. However, implementing these approaches in a suitable formal logic is often difficult, as the problems with parallel activities show. Unfortunately, few implementation details are given, but it seems likely that each timeband system must be modelled manually; there is no timeband to constraint logic compiler mentioned. It is therefore not clear that the conversion process is cost-effective.

Finally, the formalisation of timebands is taken to its logical conclusion by Wei et al. [193]. Rather than attempting to build a timebands model in an existing system of logic they instead develop a new formal logic specifically defined to implement timebands.

This new logic (called $TCSP_m$) is primarily derived from the classic Communication Sequential Processes (CSP) logic, with some similarity to the pre-existing timed CSP. The key extension over timed CSP is the 'miracle'; the non-executable top element of the ordering. This abstraction breaks some of the rules of classic CSP, but allows increased modelling flexibility (as argued in previous work by Wei et al. [192]). Much of the paper is spent defining the various properties of this new logic, with a comparatively short section on timebands themselves.

To demonstrate the modelling power of this new logic the authors turn to the classic mine-pump case study [193]. This is a classic problem from the dependable systems and formal methods fields, where the necessity to pump out water from mineshafts must be balanced against the need to maintain safe air levels (due to emissions from the pumps). This leads to the construction of a relatively simple timeband model of mine operations, although assumptions do have be made that the speed of water and carbon dioxide change can be consistently within an order of magnitude (so that events always fit in the same timeband).

Having constructed this model, the authors wish to move onto automated verification. However, the extensions to CSP (in particular, the miracle) mean that $TCSP_m$ cannot be verified by existing theorem provers or model checkers. The FDR model checker lacks support for timed CSP and theorem provers are described as too time consuming. Timed automata are

suggested as a better fit for $TCSP_m$, but the implementation required is described as a 'massive amount of work'. Instead, a manual conversion is made to a corresponding timed automata model (lacking some of the temporal precision of the timeband model) which is used with the UPPAAL model checker [105] to prove some useful safety properties. Given the complete conversion to timed automata, is appears that no direct benefits can be obtained by the use of $TCSP_m$-formalised timebands.

However, by this stage of formalisation the Timebands approach no longer demonstrates a clear socio-technical approach; it has become another new process logic, using the classic process logic examples and lacking any suitable tool support. From a social-technical perspective it appears complex and overly formal, requiring substantial formal methods skill to apply effectively. This highlights the difficulty in determining the optimal level of formality for modelling socio-technical systems; there are clear advantages in analytical power that can be obtained by increased formalisation, but this should not come at the cost of excessive modelling complexity or the lack of wide applicability.

2.13 Responsibility Modelling

2.13.1 Responsibility origins

As outlined in the introduction to this thesis, responsibility has been proposed by several authors as a basis for modelling socio-technical systems [40, 166, 179]. These techniques have links to many of the other methods discussed in this literature review. Further, Cholvy et al. [29] note that responsibility can have three different meanings (causing something (bad) to happen either by action or inaction; liability, blame or credit for an occurrence; the power to make (potentially justifiable) decisions), each of which can have greater or lesser emphasis in different responsibility modelling notations and methods.

The first and second meanings broadly correspond to causal and consequential responsibility as used elsewhere in the literature (e.g. [39]); the third meaning is more abstract, but widely used in normal language. While they are not able to completely formalise responsibility, the authors are able to further subdivide these three meanings (into concepts such as responsibility by fault, responsibility for agency etc.) and construct representations of them in deontic logic.

Feather [55] developed a system specification language called Gist. This paper is, as far as we are aware, the first use of the concept of responsibilities in requirements specification or system analysis. In Gist, systems are specified by defining all possible transitions (or deltas) within a system and then generating the 'set of acceptable histories', a set of all paths through the system that meet certain constraints. Constraints are enforced by pruning branches of the full set of histories that fail to meet the constraint. This naive version of the approach is

limited as the order of evaluation may arbitrarily limit the options of a system agent; different results can be obtained by considering agents in different order - such as in the case where two agents must not both choose the same option.

This is addressed by defining an agent or agents as responsible for a certain constraint. This responsibility means that an agent may not make any choice that will lead to a state where the constraint is violated. Unassigned constraints are implicitly treated as the responsibility of all actors. Shared responsibilities are however only weakly defined, as the logical system proposed cannot differentiate between joint responsibility (AND relationship) and separate responsibility (OR relationship) when multiple actors hold the same constraints.

Responsibilities were also used for system design by Wirfs-Brock and Wilkerson [198]; their focus was on using the concept of responsibility as the core idea when designing the structure of classes in object-oriented software design (once requirements had been agreed), rather than using them them for requirements capture itself.

The initial version of the ORDIT methodology [18] used responsibilities as an intermediate step to aid in reasoning about the more concrete elements of the approach; in particular they were used to decide between different types of relationships between agents. Strens and Dobson [178, 179] expand significantly on the use of responsibilities within the ORDIT approach. Responsibilities are identified as a potential 'boundary object' - a conceptual view of the system that can be understood in both a technical context (for system implementation) and an organisational context (for evaluation and discussion of the model). Responsibilities are treated as being held by some agent; much of the paper is devoted to the issue of handling the delegation of responsibilities between agents. This is resolved by distinguishing between functional obligations (which are linked to requirements in the system design) and organisational obligations (which arise when tasks are delegated to other agents).

Strens and Dobson [179] further expanded this structure into a three-level system model; in descending level of abstraction these are responsibility/agency, obligation/role and activity/agent views. However, the paper explicitly excludes resources and actions from this approach and instead argues that these should be modelled separately using data and process models. As a result, the approach is not able to capture certain system aspects, such as responsibilities that require resource availability to be completed.

Harper and Newman [70] proposed a simple design heuristic using responsibilities, intended as a counterpart to more elaborate methodologies such as ORDIT. They investigated system rejection, and identified clashes over perceived responsibilities to be a major cause of rejection. As part of a case study they modelled (in a semi-formal manner) the responsibilities of staff in a large multinational organisation. This model was created after ethnographic study, and identified a number of non-formalised processes that staff felt responsible for. Concurrently a new data management system was being installed, and the authors correctly predicted system

rejection by highlighting the lack of support for these perceived responsibilities in the new system. As far as we are aware this is the first work to use the concept of responsibilities for socio-technical analysis, rather than strictly for modelling.

As part of the large DIRC (Interdisciplinary Research Collaboration in Dependability) project Dobson and Sommerville [43], Dobson and Martin [41] & Dobson [40] examined the use of roles as a basis for system modelling and argue that roles are really sets of linked responsibility relationships. They consider a responsibility to be a relationship between two roles (one holding the responsibility and the other giving or issuing the responsibility) to perform some task or maintain some condition where the holder is responsible (in some way) to the giver. In this view responsibilities cannot be treated on their own, as all responsibilities must inevitably have both holders and givers.

Responsibilities can be of different types, and the authors identify two main types. Consequential responsibilities require an agent to answer for or justify some action or occurrence, while causal relationships require an agent to perform activities to realise or prevent some event. Sommerville [167] argues that it is generally not possible to delegate consequential responsibilities, while causal responsibilities often are delegated by the original holder.

Dobson and Martin argue that correctly identifying and understanding the type of responsibility is vital to prevent vulnerabilities, as wrongly defining a causal relationship as a consequential relationship can easily lead to key actions not being performed. Overall, it is argued that the one-sided view of roles leads to substandard system designs, and that better results are obtained by considering roles as relationships between two entities.

Dobson also introduces the concept of conversations, which can be used to differentiate between different roles held by the same actor. This can help to identify cases where an actor holds conflicting responsibilities (consider a doctor in general practice who holds a responsibility to treat patients, but also a responsibility to minimise the costs of his business) by treating each set of responsibilities between an actor and another entity as a discrete role.

These themes are developed in further work by Dobson and Martin [42], who produce several standard or meta-models of responsibility for common processes. These models contain subresponsibilities (e.g. allocation, procurement, dispatch in the case of resource management), agents, resources and communication channels. Particular focus is paid to processes that cross organisational boundaries, as this can easily lead to confusion or dispute over responsibility. The responsibility meta-models can be operationalised by dividing them according to actual or potential organisation boundaries; this allows analysis of potential communication problems in current situations as well as ‘what-if’ analysis of alternate options. The concept of conversations is also employed here and is used to identify a set of potential failure modes.

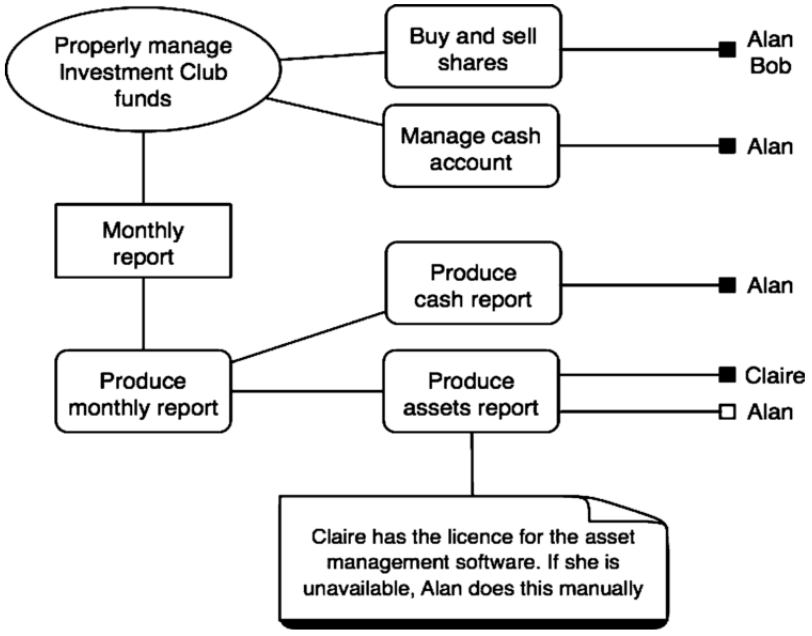


Figure 2.12: Decomposition of a responsibility, showing processes and actors (from [166]) Reproduced by permission of Springer.

2.13.2 Graphical Models of Responsibility

Sommerville [166] introduced the first methodology for explicitly modelling responsibilities. In this approach, responsibilities are decomposed into abstract goals and concrete processes, which are assigned to agents. Goals can be achieved by producing the appropriate evidence, and evidence is generated as a result of processes. These concepts are represented using a graphical notation, as demonstrated by the example in Figure 2.12.

The set of relationships between agents and processes is complex. Firstly, a distinction is drawn between causal and consequential responsibility. As well as differentiating between performing actions and justifying actions it is argued that consequential responsibility is normally associated with goals and abstract processes (e.g. a building supervisor is responsible for security), while causal responsibility is associated with concrete processes (e.g. a janitor is responsible for locking the front doors at night) and evidence. Additionally, the notation distinguishes between normal and exceptional causal responsibility, hence allowing modelling of fall-backs and emergency procedures.

Six different types of responsibility vulnerability are identified, which can be used to analyse a scenario once a responsibility model has been constructed. These include issues of unassignment, duplication of duties, uncommunicated assignment, lack of resources, overload and fragility. Some of these issues can be identified easily (assignment problems, lack of resources) while some are more complex. Fragility, for example requires an analysis of the entire model to identify which tasks are most critical, and which of these critical tasks do not

have fallback tasks or multiple agents capable of discharging them.

The authors also discuss in some detail the role of authority, in the form of some entity that is able to judge if a responsibility has been or can be discharged. They hence identify two additional vulnerabilities arising from this - lack of authority (where an agent relies on other agents they do not control) and conflicting authorities (where an agent is responsible to some agent for some process, but is generally managed by another).

Sommerville demonstrates this technique using a hospital bed management system case study. The process focuses on modelling individual, high-level responsibilities in detail, rather than attempting to link related responsibilities. Typically, a responsibility is refined into several processes with accompanying evidence. Notably, it is recommended to separately model consequential and causal responsibility relationships (to reduce clutter and complexity) and only integrate them into one version after any conflicts or inconsistencies are resolved. It is also suggested that alternative notations may be used for some stages of the modelling, such as using BPMN to construct a workflow diagram for causal responsibilities.

Sommerville [167] also proposes a pattern-based approach to responsibilities. A standard template is provided, which lists potential responsibility properties. Standard fields include descriptions of the responsibility and a description of the context; several new definitions are also used. Responsibilities can be classified in terms of their activities and their approach, giving a classification such as {Rule-based, Monitoring}. For this, three types of implementation strategy and three types of activity are defined. Other fields include pre- and post-conditions on the overall state of the system, as well as workflow models where appropriate.

Increased emphasis is also placed on modelling the workflows of causal responsibilities and several examples are stated in the BPMN notation. These models are not intended to describe specific workflows such as those of a particular organisation; they are intended to be reasonably generic, allowing them to be used in a range of situations. As such, they are proposed as another aid to thinking about the design process, rather than a design solution themselves.

Three main advantages are claimed for using this approach: contingency plans are more easily formulated; responsibility assignment is less error-prone; workflows can be combined with assignment models for vulnerability analysis. However, it is unclear how these benefits are specifically obtained by using generic patterns, rather than individually modelling relevant responsibilities. Benefits in reusability and standardisation may outweigh any lack of precision, but this argument is not made explicitly.

This form of responsibility modelling is put in practice by Ramduny-Ellis and Dix [148] who provide a worked example and a series of reflections. They broadly follow the approach outlined in Sommerville [166], although they do not make use of responsibility patterns.

The modelling itself is straightforward, but interesting side issues are raised. Once again, delegation and conflicts of responsibility are noted as complex issues, such as the conflict between producing an artifact quickly and producing it to a high quality. Of particular concern are the risks of 'buck passing' when responsibilities are delegated. It is argued that while delegation is primarily intended to be causal in practice this often amounts to the delegation of consequential responsibility as well. As a solution it is suggested that either systems are effectively subdivided to compensate for small failures, or that processes should be 'owned' throughout by specific individuals.

Interestingly, all analysis of responsibility modelling in this period focused on modelling existing systems, primarily to identify vulnerabilities. As such, the methodologies suggested may be less suitable for modelling new systems or variations of existing systems.

2.13.3 Responsibility Modelling

Storer and Lock [175] expanded previous work on responsibility modelling to produce a modelling system capable of representing a social-technical system using responsibilities as the core entity. In contrast to earlier work that focused on modelling individual responsibilities in detail this form of responsibility modelling encompasses whole systems, using responsibilities and associated agents and resources.

Responsibilities are defined as either objectives or processes. These objectives and processes are termed responsibility targets, and can be held by human or automated agents. Processes are well-defined procedures that can be followed by the assigned agent, while objectives are desirable conditions that should be maintained or reached by their assigned agents. The responsibility for processes can be freely assigned, but it is not possible to assign objectives to automated agents. When responsibilities are assigned there exists a responsibility authority, which created the responsibility and holds overall responsibility for it.

Responsibilities can be further decomposed by the agents that hold them. This creates subresponsibilities, which can be held by the original agent or later delegated. They are constraints on the decomposition process: objectives can be freely decomposed to produce subprocesses or subobjectives, but processes can only be decomposed into subprocesses.

Fundamental to this version of responsibility modelling is the idea of delegation. Initially, responsibilities are linked to the agent that created them; that agent is required to discharge the responsibility but is accountable only to itself. However, responsibilities are usually delegated to other agents. In this case another agent becomes obliged to discharge that responsibility, and is held accountable for this by the previous responsibility holder who now acts as an authority. The delegation process is restricted such that only the creator of the responsibility

can delegate it; this prevents the hierarchy of accountability being broken by an agent that delegates away responsibilities while remaining accountable for them.

Agents can be either human, technical systems or organisations; as mentioned, technical agents can only be assigned to responsibilities that can be discharged using defined processes. Agents can be structured in a hierarchy of superiors and subordinates, allowing analysis of potential organisational conflicts.

Interrelations and dependencies may exist between responsibilities, especially when fine-grained decomposition is applied. To support this, processes can be linked using *follows* relationships, stating that one process may not begin unless a prior process has already completed. Additionally, a special process subtype *decision* exists, which can be used to select the appropriate choice of process to implement depending on some condition.

More complex relations between responsibilities can be expressed using information flows or resources. Processes may exchange information artifacts by means of messages; these messages are outputted by a process to the other processes that follow it. Processes can depend on such information artifacts as inputs without which the process will not complete.

Resources provide a more general framework for treating responsibility requirements: information artifacts and agents can both be treated as specific types of resource. Resources are allocated to one or more agents to provide them with capability; no relationships between resources and responsibilities are provided, so it is unclear how the necessary set of resources can be determined for any specific process.

Sommerville et al. [168] contains a more detailed definition of a responsibility model. In this version there are only two top-level entities - Resources and Responsibilities, with agents being treated as a specialism of resources, which can themselves be further classified as organisational, technical or human.

Notably, this version also includes several homogeneous relationships that do not appear in other versions of responsibility modelling. Processes (but not objectives) can 'follow' other processes, indicating task ordering and a temporal model of activity. Agents can be subordinate to other agents, allowing for explicit representation of organisational hierarchies. Organisations can also be defined as being compositions of other agents, again allowing explicit representation of organisational structures. Additionally, several model transformations are defined - equivalences are provided for delegation, decomposition and allocation. However, while definitions are given for these relationships and transformations they are not explained in any level of detail, and the exact semantic meaning of many of them is unclear.

Lock et al. [116] provide a summary of responsibility modelling, as well as presenting a graphical notation for responsibility models. They present a simplified version of the notation, with the difference between processes and objectives removed and resources used to implement all preconditions on the discharge of responsibilities.

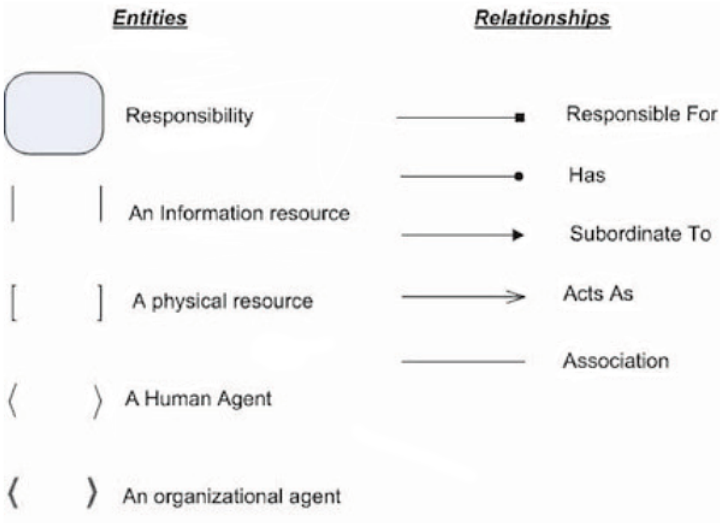


Figure 2.13: Entity and relationship types (from [116])
Reproduced by permission of Tim Storer.

This version of the responsibility model uses only three core entity types and five types of relationship, as indicated in the key to the graphical notation (Figure 2.13). All of these are drawn from the previous work [175] with the exception of the Association relationship: this relationship acts a catch-all for interactions not covered by the model and is simply a free-text description for links between entities.

HAZOPs-derived keywords are used for analysis. Some keywords are intended for analysis of specific entities (such as early/late on resources), while some appear more suited to analysing the system as a whole (such as insufficient information flow or insufficient management). Interestingly, none of the provided keywords appear suitable for analysing relationships despite the wide range of potential failures occurring in responsibility assignment or organisational hierarchies.

Additionally, the use of standard questions is recommended as an aid to the construction of responsibility models. The questions suggested focus primarily on identifying the information necessary to complete responsibilities and how this information flows between agents.

Information requirements are the main focus of Sommerville et al. [169] which highlights a number of difficulties in determining the requirements for complex information systems, especially when they are produced by assembling pre-existing (COTS) system elements. In particular, conventional requirements capture methods may identify a type of information to be recorded but fail to consider that different users of this information may require different views. For example, in a hospital it is clearly important to capture information on patient outcomes; however, the definition of ‘patient outcomes’ and exactly what information should be captured differs between medical staff and the hospital management.

To address this, responsibility modelling is recommended as a method for early stage requirements engineering, as responsibilities provide a natural notion for capturing organisational activity. An outline is given of how such models should be constructed, using a combination of document analysis (of process descriptions, business plans etc.), stakeholder interviews and ethnographic study.

Standard questions are once again recommended for considering information requirements (adopted almost directly from Lock et al. [116]) and augmented with worked examples. The examples highlight the range and also the subjectivity of this approach - some insights appear obvious from the context (emergency planning requires a map of the area) while others have been carefully refined from considering the model (late evacuation information means that police officers must be briefed in the field by radio, but radio controllers will be very busy).

The process is completed by then translating the information requirements needed to complete these responsibilities into functional requirements for an information management system. No defined process or methodology is provided for this step, but examples indicate a focus on ensuring high availability of information (degraded records are better than nothing) and stress linking these functional requirements back to their original responsibility-based purpose.

Finally, Sommerville et al. [169] conclude with a self-evaluation of responsibility modelling, arguing that it works as an effective tool for requirements elaboration and discussion with stakeholders. The intuitiveness of the concept of a 'responsibility' is argued to be accessible when discussing with system actors and stakeholders, as well as being broad enough to cover a wide range of social-technical systems and a wide range of abstraction levels. It is also suggested that responsibility modelling can be used in combination with other techniques, such as using responsibility modelling for early stage engineering before elaborating with goal- or viewpoint- based approaches.

The responsibility modelling notation was later revised by Storer and Lock [176], which provides a further simplified set of entities and semantics (similar to, but not exactly matching those in Lock et al. [116]). The modelling environment has been simplified, such as by replacing information artifacts completely by resources while the concept of process flows is eliminated entirely. Figure 2.14 shows an example responsibility model using a notation similar but not identical to that defined by Storer and Lock.

The semantics of responsibility modelling are formalised by expressing them as Z specifications that define entity types and relationships. However, the specifications do not include any dynamic rules; the behaviour of relationships under transformations is not specified. As a result, the semantics can only be enforced if the responsibility model is completely static: the behaviour of entities and relationships when the model is modified (such as by delegating a responsibility) is not defined. At this stage, the authors report that they were still attempting to formalise the approach is a consistent manner; their aim was to settle on agreed semantics

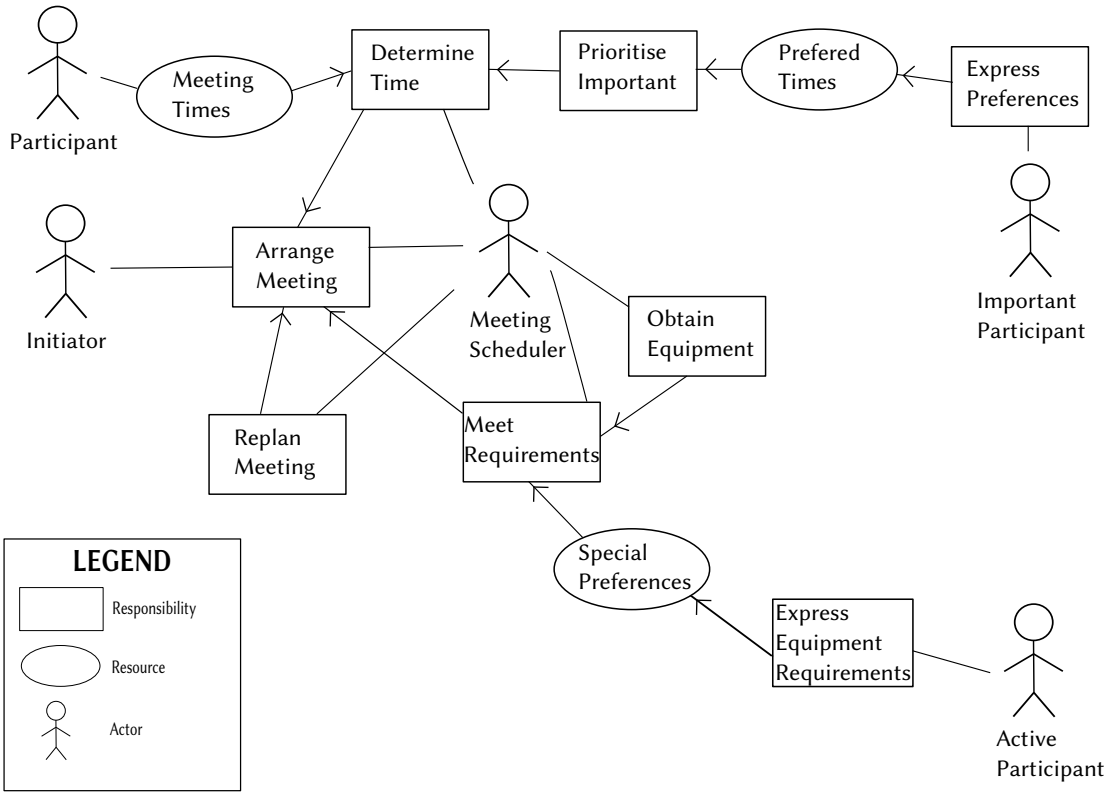


Figure 2.14: Responsibility model for the meeting scheduler described by van Lamsweerde et al. [188] using a generic notation

for a subset of the notation before moving onto more difficult areas.

While most of the paper focuses on redefining and streamlining existing concepts the issue of shared responsibilities is significantly expanded. Responsibilities assigned to multiple agents can be explicitly serial or parallel; either requiring one agent out of several or requiring full and joint action. Serial assignments can be further specified by defining the conditions for changing from primary to secondary agents - such handovers of responsibility can occur due to unavailability, overload or privilege escalation.

Several case studies and investigations were carried out using graphical responsibility modelling. Sommerville et al. [170] investigated flood management schemes in Northern England, which were under review after a series of large floods several years earlier. Unexpectedly, floods quickly reached important civic buildings such as the fire and police headquarters, which required fallbacks to offsite control and impeded communications. Sommerville et al. focus primarily on the evacuation processes during this incident - this involves escorting people from areas at risk of flooding to safer ground (in contrast to search and rescue, which involves removing those already in danger).

By creating small responsibility models they identified important information resources that were shared by multiple responsibilities assigned to multiple actors but were not kept

updated and synchronised. As a result, actions taken by one organisation to aid another (e.g. ambulances dispatched to assist the fire brigade) were unsuccessful, as they were directed to the wrong locations. Responsibility analysis of the high-level evacuation plan also noted the existence of a fundamental responsibility 'Collect Evacuee Information' that was not formally assigned to any agents. While clearly vital to the plan, all organisations involved assumed that this information would already be available without considering how it would be gathered.

Lock et al. [115] apply similar techniques to the 2007 Scottish elections, which were notably the first major UK elections to use an e-counting system. This involved paper ballots being filled out by voters which where then optically scanned and counted using recognition software. By constructing responsibility models they were able to identify a number of vulnerabilities and social issues, many of which had actually occurred in practice.

Analysis of the resources involved identified an interesting scenario where agents that did not have any direct links were in fact utilising the same resource. The optical scanning machines were being maintained (and often modified) by supplier's technicians; meanwhile, observers of the political parties were observing the machines and trying to produce an unofficial count of votes. The technicians would therefore focus on their basic role of keeping the machines operational, without realising this could raise concerns with the observers that the counting was being manipulated.

Hazard keyword analysis also identified problems with responsibilities being discharged too quickly. As the e-counting system was new, local authority staff had trained in its usage, and become quite proficient. In contrast, party officials had almost no experience of the system and so took much more time to understand its outputs. As a result, problems arose in the adjudication process, where the meaning of unclear votes is decided. Council staff quickly determined the meaning of these votes and processed them through the system, often before party officials had an opportunity to examine the ballots themselves and potentially challenge their intent. This problem was exacerbated by the system design that meant it was it was only possible to return to the previous ballot, and not go any further back.

Generally, responsibility modelling at this time was focused on arguing for the advantages of responsibilities as a modelling concept and creating an initial fully-featured modelling language. While broad in terms of concepts, this notation was not consistent and never fully formalised; different papers presented different views and different uses for the approach. This lack of formality limited strong methodological approaches in case studies: useful insights were generally obtained, but much of this may well have been possible with domain knowledge alone.

2.14 Conclusion

This chapter examined different techniques that can be applied to socio-technical modelling and analysis. These techniques come from a wide range of backgrounds, ranging from formal methods and safety analysis to organisational analysis and dependency engineering.

The techniques face a number of different challenges. Some focus on addressing the scale of modern systems, addressing large organisations and inter-operating systems. Some handle complexity - the need for precise definitions of complex, specific events and process. Others address the need for abstraction - methods that are broad enough to cover the important aspects of a system without being restricted by unnecessary or unknown details. More still target the reasoning behind systems, focusing less on how systems operate and more on why they are set up in their current form.

The broad range of techniques also highlight the strengths and weaknesses of particular approaches. Formal methods offer rigorous checks and analysis that other approaches cannot match, but their complexity can make them expensive to produce and unintelligible to many users. Requirements engineering and business process models provide flexible and often intuitive approaches, but with limited analytical power. Similarly, the inherent flexibility of holistic approaches such as SSM is both a strength and a weakness, allowing wide applicability while causing severe subjectivity in analysis. Safety analysis techniques show well-structured approaches to analysis that don't require the complexity of formal methods, but their applicability outside of specific domains is unclear. Throughout all these methods there is a clear lack of effective validation and testing, with limited evidence of the effectiveness of many techniques.

As a result, different methodologies are suited to different applications and different problem domains. Technical systems and well-defined process are suitable for more formal methods, where the extra rigour can deliver useful analysis. Systems with uncertain behaviour benefit more from more abstract methods, which capture and model the relevant elements without the assumptions and over-precision needed by other techniques. There is no unifying methodology of socio-technical modelling - the choice of methodology is dependent on the problem domain, the modeller's knowledge of the system and the intended outputs of the modelling and analysis effort.

While no one technique can be universally applied, responsibility modelling does offer a methodology with wide application. The concept of responsibility is abstract enough to capture broad influences such as social pressures and legislation but can also be used to intuitively capture details of low-level processes. This combination of abstract and concrete responsibilities also allows effective modelling of the reasoning and justification for particular system choices. However, the technique is currently informal, without a fully defined

semantics; this limits reproducibility and prevents tool support and formal checks or analysis.

Chapter 3

Research Methodology

3.1 Introduction

The scale and complexity of socio-technical systems has significant implications for their study and evaluation. The characteristics of socio-technical systems, as described at the start of Chapter 2, mean that it is rarely possible to perform controlled and reproducible experiments in the classic scientific style [156] as the resources required to construct and test multiple variations of a large system are beyond the reach of most research projects. Additionally, it is difficult to separate out the benefits gained from the experience of constructing a particular system from the benefits of applying any methodology. Attempting to control specific variables is also difficult, as socio-technical projects cannot usually be isolated from the context they operate in. Therefore a tension exists in socio-technical research: in order for a project to be realistic and representative of the real world it should ideally be an existing project that is subject to scientific observation, but in order for a project to be adequately protected from confounding variables it should be a specially constructed and controlled study.

These problems all apply to socio-technical modelling and analysis. Models cannot be effectively verified or evaluated if the socio-technical system cannot be accurately measured or subjected to testing and examination; predictions made or issues identified by analysis cannot simply be verified by changing the system and observing the resultant behaviour.

Popper [143] suggests that attempting to prove a model or theory as universally true is unproductive, and that instead validation efforts should focus on increasing confidence in the model by a series of empirical tests.

This argument may initially seem to only apply to individual models, rather than to the more general problem of modelling techniques. However, the two are very tightly coupled. Firstly, a Popperian view of modelling techniques can see individual socio-technical models as the

outputs of a socio-technical modelling technique, and so evaluating the models provides an empirical test of the modelling technique. Secondly, evaluating techniques in the abstract is difficult, and so many studies indirectly evaluate techniques either by comparing them to others, or by providing demonstration or case studies in the form of individual models.

Naylor et al. [132] aim to combine these varying stances on modelling validation by suggesting a three-step process of verification. Firstly, a list of hypotheses about the system that are represented in the model must be constructed, using the best information available to the researcher. Secondly, these hypotheses should be individually tested in an attempt to verify them as well as possible. For many hypotheses this may be difficult due to a lack of empirical evidence, but it is not necessary to abandon hypotheses merely because they cannot be tested. Thirdly, the ability of the model to predict the behaviour of the modelled system must be tested. This may consist of making forecasts of future predicted behaviour, or by comparing the predictions of the model from previous information against the actual outcomes. In particular, they stress the use of good statistical tests on numerical data.

An accompanying critique emphasises the importance of understanding the purpose of any specific model, as simulation models are generally used when other techniques are not suitable, such as when modelling interactions between many processes. This makes the analysis more specific - does the model fulfil the purpose for which it was created? If it doesn't, the underlying hypotheses are refined, and so the modelling and verification steps are combined as an iterative process.

These challenges appear in most forms of research attempting to understand complex structures of human organisation and interaction; there are significant similarities to many areas of the social sciences, and socio-technical systems thinking itself emerged as a social science framework or philosophy. This leads researchers to mainly apply qualitative methods such as case studies and interviews, rather than attempt to construct complex quantitative experiments. These similarities also lead to the common adoption of research approaches and frameworks that are inspired by other disciplines outside the natural sciences, such as action research [197] and design science research [141].

It can also be difficult to determine which metrics define a 'good' socio-technical model or modelling technique. Accuracy may appear to be the most obvious property that defines a good socio-technical model, but accuracy can be difficult to judge - for example, is it the representation of the system in the model or the results generated from the model that should be tested for accuracy? It is also not clear exactly what the accuracy of the model would be tested against, as each instance of a socio-technical problem has subtly different factors and implications. An alternative approach is to assess models on some usefulness function, attempting to judge how well they assist in the performance of some analysis, improvement or validation task. This reflects the multiple uses for which different models and modelling

techniques may be applied (for example, the differences between using a model for risk analysis and using it for process improvement), but at the cost of generality as the usefulness of the model in another context cannot be easily inferred. Assessing in terms of usefulness is also subjective, as it reflects the intentions of the model user; for example, a model that identifies security risks that require greater investment is unlikely to considered useful by a manager who wishes to cut costs, even if the risk identification is accurate. This makes effective evaluation of socio-technical models and techniques difficult - the purpose, criteria and metrics that they should achieve are all varied or unclear.

This chapter explains the research methodology of this thesis, in the context of research methodologies in socio-technical systems in general. This encompasses research structure as well as individual research and evaluation techniques. The remainder of this chapter is structured as follows. Section 3.2 surveys the research methodologies and validation techniques of the papers previously studied in our literature review and identifies a consistent lack of methodological validation techniques. Section 3.3 discusses metrics for assessing socio-technical methodologies - what characteristics make a good model or a useful technique?

Section 3.4 considers research philosophies and their application to socio-technical study; in particular, the arguments for against positivism and realism are explored. Section 3.5 examines two research frameworks that often are applied in information science and socio-technical modelling - design science and action research, and explores the differences and similarities between them. Section 3.6 discusses case studies and interviews - two of the most common validation methods in information modelling, with a particular focus on their application to socio-technical problems. Section 3.7 then provides a comprehensive description and discussion of the case study-based validation methodology that we apply in Chapters 6 and 7. Finally, Section 3.8 provides an overview of the chapter.

3.2 Evaluation approaches in socio-technical analysis

A survey of the evaluation strategies described for each of the methods reviewed in Chapter 2 was undertaken. The purpose of the survey was to establish which methods were used in practice for evaluating socio-technical modelling methods. The results of this survey were used to inform the selection of an evaluation strategy for the current research. An overview of the socio-technical methods examined and evaluation strategies used is provided in Table 3.1.

Yu's original paper on i* does not include an evaluation of the technique [203], although reference is made to a case study by Briand et al. [22] using a variation of the actor-dependency notation. This involved a study of maintenance processes at the NASA Software Engineering

Modelling Method	*Main Evaluation Strategy*
i*	Case studies
KAOS	Worked examples
Responsibility Modelling	Case studies with comparison
MORT	Collection of case studies
HAZOPS	Extensive use / Comparisons to other techniques
STAMP	Worked examples

Figure 3.1: Modelling methods and evaluation strategies

Laboratory, using an existing audit methodology developed by Briand et al. [21] and adapted to use actor-dependency models. This provides a structured approach for examining the maintenance processes, but the construction of the actor-dependency model does not appear to follow any defined methodology. Validation of the model was performed by interactions with stakeholders during the process improvement process. Similarly, the extension of i* to model trust by Yu and Liu [204] demonstrates the new features using examples drawn from a case study. However, the efficacy of neither the model or the extension are evaluated. Furthermore, there is no structured evaluation of the methodology in papers discussing i* security analysis [112], e-service design [68] or in Yu's own thesis [202], which all use case studies for demonstration.

No validation is described in the original KAOS paper [36] of either the applicability or usefulness of the notation, although a well-structured methodology for using the technique is presented. The classic KAOS meeting scheduler case study [189] mentions the use of operational scenarios for validation purposes, but gives no real details of how these were applied. They note several elements of the problem domain that they were not able to model, without a discussion of how that effects the accuracy of the model. Extensions of KAOS such as mappings to BPM [101] and to VDM [131] are also unvalidated. An application of KAOS to scalability requirements [47] provides some evaluation of the technique based on comments from domain experts, but does not appear to use a structured methodology.

Sommerville et al. [169] present a set of criteria for evaluating requirements engineering techniques and use these to informally assess their responsibility modelling method. Elsewhere responsibility modelling is either presented without reference to validation [175] or is demonstrated using case studies that are validated against system operation and previous events [170].

In the field of safety-critical systems the need for validation is more apparent. The MORT [94] method drew on existing knowledge from fault trees techniques, and was evaluated through a collection of case studies of previous accidents. The evaluation compared the set of contributing factors to accidents identified through MORT with those identified by existing techniques used in accident reports and found that MORT detected all contributing factors

identified by previous methods as well as detecting additional contributory factors. However, the details of the specific case studies and existing techniques chosen for comparison are not stated, which obstructs reproducibility and potentially reduces the generalisability of the study. The HAZOPs method has been tested by decades of application in the industrial sector [182]; Dunjo et al. [48] report in their HAZOPs literature survey that validation of HAZOPs was mainly carried out by comparison with other techniques and the reporting of industrial experience. For example, Hoepffner [79] compared HAZOPs to fault trees and FMEA, while Sweeney [183] provides insights from many years of experience applying HAZOPs in a chemical company.

STAMP [109] is a safety analysis technique drawing on system theory; the original paper demonstrates the technique and notation through a case study, but does not validate either the case study or the technique itself - occasional references are made to a NASA incident report, but these are primarily used as a data source, rather than as validation. FRAM likewise provides demonstration of the technique through worked examples [80], but does not validate it. This lack of validation in systematic accident models was the subject of a study by Underwood and Waterson [186]. They study the STAMP, FRAM and Accimap techniques, and identify a lack of formal validation and evaluation of usability and reliability across all three techniques. They note the use of case studies to obtain empirical validation of systemic accident models, but state that this is still 'far from extensive'.

The lack of a consistent approach to model and modelling validation is clear; many models and techniques are not validated at all, while case study demonstration and validation is by far the most common method used to establish confidence in models and techniques. Different variations of case study validations are used, ranging from demonstrations of the technique, validation based on the modeller's own experience and to more elaborate studies where models are produced and then validated against previous system deployments and incidents or are compared to the results of other, more established analysis techniques.

None of the studies cited directly attempt to argue the validity of their technique from first principles, instead relying on more indirect forms of proof or confidence. This structure broadly corresponds to the three-step process of Naylor et al. [132] : firstly a general underlying framework or philosophy for the technique is constructed, generally using existing research (such as systems theory in STAMP, or the concept of responsibilities in responsibility modelling); secondly, a general argument is made as to why that underlying concept is useful (such as the emergent properties of system theory, or the intuitiveness of responsibilities); thirdly, examples and case studies are used to show the technique can be used effectively.

These studies contain many subjective elements that limit confidence in the correctness of the models or techniques; in particular, the choice of case studies can hide the weakness of methods in particular areas, and few methods are systematically deployed by modellers

other than the method's creators. However, validation and verification in the socio-technical space is an inherently difficult problem and incremental increases in confidence are the best validation strategy available.

3.3 Criteria & Metrics

Baxter and Sommerville [11] note that socio-technical methods are rarely evaluated for their efficacy, and that reports of successful use are 'comparatively scarce'. They also argue that an important barrier to assessing the success of socio-technical methods and projects is a lack of suitable evaluation criteria, especially considering the social aspects of systems, and that the wide range of stakeholders leads to multiple different viewpoints on 'success'.

Sargent [154] provides a contemporary overview of verification and validation methods for simulation models. They describe a wide range of validation techniques, which vary in complexity from simple visual displays to complex statistical tests. Validation techniques that are suitable for socio-technical problems include:

- Comparison to other models - the results and outputs are compared to those of other models that are already accepted as valid
- Event validity - Simulating events in the model delivers similar results to real-world events
- Face validity - asking knowledgeable system experts to assess the reasonability and accuracy of the model
- Historical data collection - historical results (that were not used to design the model) can be used to test model predictions
- Predictive validation - the model is used to forecast system behaviour, which can then be observed
- Turing Tests - experts are presented with a scenario and two different results, and asked to determine which result comes from the model and which from the real system

Not all techniques can be applied to particular socio-technical problems. In particular, socio-technical models may focus on the behaviour of a system under specific circumstances which may not regularly occur in normal operation (indeed, analysing such unusual circumstances is a common use case for socio-technical modelling). This severely limits the use of predictive validation and reduces the amount of data available for historical predictions. Expert-based

analyses such as face validity checks are more widely applicable as long as access to experts is available, and model comparison is highly useful in well-studied domains.

Sommerville et al. [169] provide a list of potential criteria for evaluating requirements engineering methods as part of their work on responsibility modelling, although their technique overlaps heavily with socio-technical analysis. These are:

- Naturalness: Can stakeholders (without experience of the technique) understand the models?
- Applicability: Can the method be applied across a wide range of domains?
- Scalability: Can the approach scale to model large, real-world systems?
- User involvement: Have end-users been involved in the development of the notation and the models?
- Complementarity: Does the method compliment other techniques?

None of these criteria directly address either the ability of a technique to produce accurate representations or the usefulness of models for a particular intended purpose; they address secondary characteristics of the technique such as usability and breadth. However, there are clear advantages from using techniques that meet these criteria. For example, scalability is vital for any model intending to capture a large-scale system, while complementary methods are necessary for any technique being deployed as a part of a larger project lifecycle. Conversely, a method meeting very few or none of these criteria is unlikely to be of practical use, regardless of the intended purpose.

Costello and Dar-Biau [34] suggest a set of metrics for requirements engineering based on a lifecycle approach, addressing such issues as volatility of requirements, traceability of requirements and detection of defects. These metrics are intended for assessing full requirements engineering processes, rather than modelling notations or single-stage methodologies and so their applicability to the specific area of socio-technical models is somewhat limited. However, it is clear that for a socio-technical modelling technique to be of use in requirements engineering it should support the overall process in meeting these types of metrics.

Three of these metrics are particularly relevant to socio-technical modelling. Requirements traceability requires the ability to trace individual requirements to both their high-level rationale and their lower-level specialisations or implementations. Socio-technical modelling techniques can support this by providing explicit mechanisms for relationships such as decomposition, specialisation and abstraction, and by enabling references to source documents. Requirements completeness requires the ability to determine and specify requirements at multiple levels of detail, and to indicate when the appropriate level of abstraction has been

reached. This requires modelling techniques that are capable of handling these multiple levels of abstraction and that can easily match high- and low- level details. Finally, integrated progress requires processes and tools that can operate collaboratively with common structure and terminology; in socio-technical modelling terms this corresponds to Sommerville et al. [169]'s 'Complementarity'.

3.4 Research Philosophy

Philosophy of science is concerned with the underpinning foundations of science, such as defining what activities can formally be defined as science, analysing different forms of scientific reasoning and considering the nature of scientific truth. Many of these topics are still subject to significant debate amongst philosophers without a clear resolution, and generally not considered relevant to the day-to-day conduct of scientific research [136]. However, there are two main areas of research philosophy that are particularly relevant to the study of socio-technical systems, and that may influence a researcher's choice of approach and methods. These are the debates around positivism, especially outside of the natural sciences (often contrasted with interpretivism) and the distinction between realist and anti-realist interpretations of scientific theories.

Positivism is a school of thought that argues that (completely certain) knowledge can only be based on natural phenomena and hence that observations of such phenomena form the only valid basis for scientific knowledge. In this sense, positivism draws from empiricism, where scientific observations are the only acceptable base for scientific theories. As such, they reject the role of intuition and a priori (based on entirely on logical reasoning) arguments. Positivism and its variants are implicitly are accepted and applied in most scientific research. However, classical positivism has several vulnerabilities. For example, mathematical proofs do not rely on any observations of experience; they are purely a priori arguments. Positivism would therefore imply that mathematical proofs are not scientific, but mathematics is almost universally accepted as a science.

In philosophy of science, critiques of positivism have lead to the development of new theories, such as post-positivism which attempt to retain the desirable features of positivism while acknowledging its original shortcomings. Post-positivism acknowledges that scientific research is not truly neutral, and that characteristics of the research or researcher can influence the observed evidence. An important contribution is Popper's [143] argument in favour of falsification as opposed to verifiability, which addresses the problem of theorising about unobservable or unmeasurable phenomena. Post-positivists still favour obtaining scientific evidence by conventional experimentation, but acknowledge that other approaches may also be valid.

In the applied and social sciences positivism can be considered to be opposed by anti-positivism, which critiques the use of the classic scientific method in these fields. This suggests that social behaviour is inherently subjective, and can only be understood in conjunction with theories and concepts. This makes it impossible to collect scientific evidence and then construct a theory that explains this behaviour; the evidence can only be understood and analysed in a certain context.

A related debate surrounds the topic of realism - whether or not scientific theories attempt to explain the objective truth of their subjects, or instead should be treated only as effective predictions. In particular, this debate applies most strongly where certain aspects of a scientific theory are not directly observable. For example, in the Standard Model of particular physics protons and neutrons are formed of quarks; elementary particles that cannot exist independently. A realist interpretation of the Standard Model argues that particles really are made of quarks, even though this cannot be directly observed; an anti-realist interpretation suggests that the theory of quarks is simply a mathematical model that very accurately predicts the behaviour of particles - it is irrelevant as to whether this is 'true', as long as the model proves effective in prediction.

The realism debate ranges across a wide spectrum. Many definitions of realism admit the possibility that scientific theories may not be objectively true; for example, it is now clear that classical models of the atom such as the 'plum pudding' model are false, and that there is no reason not to believe that currently accepted true theories will become obsolete in the future. However, most realists would argue that the development of an objectively true theory is possible, even if present theories are not. An extreme form of anti-realism, instrumentalism, argues that there can be no truth beyond what is observable [136].

Particularly relevant to socio-technical studies is the recent concept of model-dependent realism, as stated by Hawking and Mlodinow [72]. Considering the increased use of scientific modelling, they argue that reality is considered by using observations to construct rules and theories, which construct models. Situations may arise where multiple models accurately describe the same phenomena; for example, both Newtonian and Einsteinian physics accurately describe low-speed motion. Conversely, a model may not accurately match all observations within its applicability; for example, the Standard Model predicts that the universe should consist of equal amounts of matter and anti-matter, but the observable universe is believed to consist disproportionately of matter.

Hawking and Mlodinow argue that it if there are multiple models that agree with observations, then there is no purpose to examining which is objectively true. The idea of objective truth is not completely rejected, but it is acknowledged that it may be impossible to determine. Therefore, scientific models should be judged on how accurately they predict phenomena and match existing observations, without considering whether they are 'true'.

A certain level of anti-realism is implicit in most socio-technical research. A socio-technical model (such as a responsibility model or a STAMP instantiation) is clearly not intended to be a literally true depiction of a system; it is an abstraction constructed to gain an understanding, present the important features or enable an analysis. In socio-technical analysis the aim is to make accurate and useful predictions, such as determining the potential causes of a failure or the performance of a sub-system. The principles of this analysis are based ultimately on knowledge obtained from observation; for example, an abstract concept such as the overload of a system is derived from observations that systems often fail when handling an excessive number of tasks. These principles should then hold predictive power, and further studies can test whether or not the predictions are in correspondence with the observed behaviour of a wider range of system.

Overall, most socio-technical and information science research can be considered to adopt a broadly post-positivist approach. The scientific method is adopted, and the interpretivist, subjective approach is broadly rejected. Individual studies or observations can certainly be subject to bias and subjectivity, and different interpretations can produce different results. However, a synthesis of studies can still meet the scientific standard. An approach similar to model-based realism is implicit; models and rules for behaviour are constructed and tested, and it is acknowledged that multiple techniques can provide similar predictive power, although certain techniques may be more effective in certain domains or when considering certain behaviours.

This philosophy has implications for the choice of research methods in socio-technical study. The level of interpretation required should be strictly limited; this encourages the use of experiments where possible and case studies (performed in an objective way, focusing on the observed facts) where not, alongside the use of simulations and model predictions [63]. Methods that grant the researcher subjectivity should be avoided. The difficulty of experimentalism in socio-technical systems of non-trivial size has already been discussed, so techniques that allow the impartial observation of socio-technical systems (such as participant interviews, direct observation, case studies through document analysis etc.) are optimal.

3.5 Research Frameworks

As discussed earlier, the conventional experimentation-based scientific method is usually impractical for studying socio-technical systems, due their scale and complexity. Instead, other research frameworks offer principles and methodologies that are more suited to research where elements of the system cannot be controlled, and where the process of performing a study is itself part of the research outcomes. Two particularly relevant frameworks are action research [123] (a dynamic approach that blends the roles of researcher and research

participant) and design science [141] (a methodology that focuses on the construction and evaluation of a specific artifact).

3.5.1 Action Research

Action research is a research framework that combines the roles of researcher and participant; research is carried out about some problem, while the problem is actively addressed as the research is ongoing. As a consequence the nature of the problem may change during the duration of the research. In particular, it is often used by professionals for investigation and improvement of their own working practices [134]. This breaks down the traditional divide between research (attempting to understand the problem) and action (trying to fix the problem, usually based on a body of research-based evidence) [197]. Inquisitive research is performed with the direct intent of causing change using an iterative cycle such as Observe, Reflect, Plan, Act, Evaluate [123].

This leads to a dynamic structure of research where ideas are quickly generated and then tested by applying them directly to the problem and analysing the resulting changes. Action research is not tied to any specific set of methodological techniques; different research methods and techniques can be applied within the framework of action research. Action research is by definition highly subjective and often unreproducible; defences to the validity of action research range from the use of 'validation meetings' with other participants/researchers [123] to Marxist theory rejections of universal truths [197]. Action research (particularly in the context of information systems) functions similarly to consulting, which raises practical and ethical issues [9].

This form of 'classical' action research is the most common in information systems research. Supporters of action research such as Baskerville and Wood-Harper [10] make strong arguments for its widespread use: 'Where a specific new methodology or an improvement to a methodology is being studied, the action research method may be the only relevant method presently available'. However, the acceptance of action research in computing is still widely debated [134]. 'New action research' is common in other fields, and places greater emphasis on the individual and moral outcomes of the research process, and less on delivering broader contributions to the body of knowledge [134].

There is a long history of action research being linked with socio-technical systems - Mumford [129] states 'The story of socio-technical design is closely allied with action research', although this link is more philosophical (both action research and early socio-technical studies were concerned with a participatory approach to change, especially in industry) rather than implying consistent use as a methodology.

3.5.2 Design Science

Oates [134] defines the 'Design and Creation' research strategy (also known as design science), where the purpose of the research is to create an *artifact*. Artifacts can be systems such as computer programs, but also instances such as system models. Additionally, methods and methodologies can themselves be treated as artifacts and the development of methodologies falls under the design and creation paradigm. A typical design science project may consist of five iterative steps:

- Awareness: the recognition of a problem or opportunity to be addressed
- Suggestion: the leap to having an initial idea to address the problem
- Development: the implementation of of the idea and construct of an artifact
- Evaluation: the assessment of the value and worth of the artifact
- Conclusion: the consolidation of the research, and examination of unexpected results

Similar structures are used throughout design science; Peffers et al. [141] study seven different papers that propose broadly comparable processes and objectives, before constructing their own process that consists of six steps: Problem Identification, Solution Objectives, Design and Development, Demonstration, Evaluation and Communication. Notably, their process differs by suggesting that research need not start at the first of these steps - for example, observing a useful solution in practical deployment may lead to a post-hoc analysis of its origins and development process. They also observe a surprising lack of application of design science in information science domains such as requirements engineering, which they attribute at least partially to the lack of well-defined methodologies.

In design science, the evaluation of the artifact can be contentious, and varies between disciplines. In computing science, artifacts may not be subject to rigorous evaluation - the aim is demonstrate a proof of concept that justifies the initial idea, and the testing of the proof of concept may not be considered interesting. Alternatively, 'proof by demonstration' evaluation involves limited evaluation of the artifact, where a suitable evaluation technique is used but on a limited scale. For example, a model may be tested using a small example that is not as complex as real-world scenarios, or evaluated by academics and students rather than actual users. In contrast, information science researchers place more importance on real-world evaluation, such as case studies or action research.

Despite coming from different backgrounds there are clear similarities between action research and design science. Jarvinen [92] compares the fundamental characteristics of design science and action research, and notes a very high similarity between both approaches. For example,

both focus on action (or building) and evaluation, both generate knowledge that benefits practice and both focus on assessing products and changes in terms of utility. In contrast, Iivari and Venable [85] label action research and design science as 'decisively dissimilar'. They note the significant ontological and epistemological complexity within action research, which is discussed less frequently in regards to design science. They also note that action research is often carried out in order to understand existing reality, while design science by definition seeks to produce new and innovative solutions. While there is significant scope for action research and design science to be used together for mutual benefit, they conclude that many similarities are superficial and they remain clearly distinct approaches.

3.6 Research Methods

Evaluation of models and modelling techniques require evidence to assess the accuracy or utility of the model or techniques. The most common way to obtain such evidence in socio-technical modelling is to conduct a case study - an examination or application in a specific problem domain. Numerous methods can be used to gather information for a case study, including analysis of records and documents, direct observation of the system and interviews with system participants and stakeholders. Interviews are particularly notable as they can be used both to gather information for a case study and for evaluating case studies, such as by presenting models to domain experts.

3.6.1 Case Studies

Yin [200] defines a case study as an inquiry based on two main points:

- A case study investigates a complex contemporary phenomena within its real-life context, especially when the boundaries between phenomena and context are not clearly evident.
- A case study copes with the distinctive situation where there will be many more variables of interest than data points, and hence relies on multiple sources of evidence with data needed to converge in a triangulating fashion.

In particular, case studies are useful where 'the investigator has little control over events' [200] and where an experimental approach is impractical, which fits well with the scale problems of socio-technical analysis. Yin writes with regard to the use of case studies within the social sciences, where the aim is to investigate some scenario, and either explore, describe or explain it. Oates [134] notes that in computing science disciplines the term 'case study' is

often used to mean the application of a particular technique or method to a particular scenario, often not representative of real-world activities. This is in contrast to the more widespread use of case study research, which examines the complexities of real-world issues. Many case studies in requirements engineering and systems analysis fit this narrower definition, where a model of a system is produced without attempting to compare it to real-world practice. Instead, the production of the model (often accompanied by some form of analysis of vulnerabilities, design options or a discussion about the application of the technique) is intended to demonstrate the technique and show it can be applied in that particular context.

Oates [134] further distinguishes between three types of case studies:

- Exploratory case studies, which are used initially to identify areas of research for further study
- Descriptive case studies, which provided detailed analysis and a narrative around the context of the study
- Explanatory case studies, which seek to identify the causes and factors that cause particular outcomes

Critiques of case study research contend that case studies cannot be generalised beyond a single example, and state that they should be used only as the initial part of a study [1]. Flyvbjerg [61] aims to rebuke this and several other common misconceptions about case studies. He emphasises that single case studies are perfectly capable of falsifying pre-existing theories, and that case studies contribute to widening the body of scientific knowledge. Flyvbjerg also attempts to rebut the potential subjective bias of case studies by demonstrating that published case studies tend to focus on falsification, rather than validation. Ragin [147] notes that small-scale studies provide the flexibility to reconsider and adapt based on the experience of the study, which may be lost in larger research.

A common feature in case studies is the use of ‘archival analysis’ [200] , where analysis is performed by studying existing documents and data sources without directly observing events or communicating with the individuals involved. Case studies may use archival analysis techniques for a significant proportion of their content - this may range from the use of documentation to define initial hypothesis and points of interest, or documentation may be used to construct an elaborate world view or model that is then examined by participants or compared to direct observations. Oates [134] notes the advantages of document analysis as information can be obtained quickly and easily, and can provide information that would not otherwise be available (for example, historical details). However, official documents can be particularly vulnerable to large differences between documentation and reality. Wohlin et al. [199] notes that archival data consists of materials that were not initially intended for

research use and may be of variable quality, although more direct forms of information also have disadvantages.

There are also overlaps between case study research and action research - both involve a careful examination of a particular system, generally with the hope of generalising results more widely. Action research differs from case study research in that action research also involves the process of making change within the system as part of the research process, while case studies are purely observational [199].

3.6.2 Interviews

In interview-based data collection participants are asked a series of questions about the area of research. Tightly structured interviews (like face-to-face surveys) may collect primarily quantitative results, but interviews can also be primarily qualitative in nature. This allows them to generate 'rich' or 'compelling' data drawing on the personal experiences of the participant, and is ideal for illustrative examples [66]. In particular, this can capture subtle nuances that would not be detected by bulk collection techniques, and can be used to understand why individuals act or feel in a particular way (rather than just how they act). However, care must be taken that individual accounts are not overgeneralised or assumed to represent an entire group.

Interviews are classically grouped into three types: open, semi-structured and structured, referring to the level of pre-planned structure enforced on the interview by the researcher. In case study research it is common to use semi-structured interviews in preference to open or structured interviews [199]. According to Gillham [66], the semi-structured interview has these defining characteristics:

- the same questions are asked to all participants
- the form and type of questions are developed to ensure focus
- prompts are used to cover areas that are not spontaneously addressed
- participants are interviewed for broadly the same time
- questions are open, without leading the participant
- probes are used to elicit additional comment from the participant

Semi-structured interviews generally consist of a series of open questions that cover the relevant areas of interest, combined with short prompts that correspond to sub-areas covered by that question. Participants may naturally cover all areas of interest in their response to the

main question; if they do not, the interviewer uses these prompts to redirect the participant to the missing sub-area. This requires the interviewer to have evaluated the relevant areas in preparation before the interview in order to ensure the correct coverage, ideally by means of a pilot study. Analysis of the interview results is performed after transcription.

Semi-structured interviews provide a balance between the breadth of open interviews and the consistency of structured interviews; this can provide the benefits of easier analysis and consistent coverage associated with structured interviews, while retaining much of the richness of unstructured interviews. However, they require a significant time investment in both interview development (question and focus) and post-interview analysis (transcription and evaluation) and can be reliant on the skill and experience of the interviewer to obtain useful results.

3.7 Evaluation Methodology

The primary contribution in this thesis is the development of a formalised responsibility technique, with an updated notation and significantly expanded automatic and semi-automatic analysis techniques. We evaluate the extent of this contribution through two case studies and one explorative study of modellers. These experiments are used to assess the extent to which formalisation eases the modelling process and provides more consistent results. The focus of these studies is to identify the strengths and weaknesses of the formalised responsibility modelling notation and the related analysis techniques. These are both intrinsic features of responsibility modelling, and represent both its most significant elements and the hardest elements to adapt if they are proven unsuitable. In contrast, the procedure and style for constructing models is only of secondary importance, as this can be substantially modified without changing the core of the technique (and will vary between individual modellers).

Several areas of the modelling and analysis techniques are particularly important, and these are the primary focus of our studies. The modelling notation should be flexible enough to capture the full range of socio-technical behaviour without becoming overly convoluted or unintuitive. However, it should also be powerful enough to capture the detail of complex responsibility structures where necessary. Similarly, they should also be relatively easy to comprehend and construct without requiring a full understanding of the underlying semantics. The analysis techniques need to present consistent results that broadly correspond to real-world problems or that are supported by domain-specific analysis methodologies; these results should also be able to be interpreted effectively without requiring excessive domain knowledge.

3.7.1 Research Strategy

Our evaluation of formalised responsibility modelling consists of three studies - two case studies (featuring the modelling and evaluation of a specific domain) and one hybrid case study / empirical evaluation (participants model a specific domain, and then discuss and evaluate responsibility modelling as a technique). Each study evaluates particular characteristics and applications of responsibility modelling, while also demonstrating its use in a specific domain.

The first case study re-examines a problem previously modelled by the InDeED consortium using a previous version of responsibility modelling - the Hunterston Nuclear Power Stations Off-Site emergency plan. The primary aim of this study is to compare the results of formalised responsibility modelling against the existing state-of-the-art, and demonstrate that the revised technique is at least as effective. Additionally, we show that the extensions in our technique (in particular, the automatic analysis techniques) provide additional insight that earlier responsibility modelling did not.

The second case study models a specific problem domain (the TCAS aircraft collision avoidance system) and then examines the resulting model with domain experts. This study has two main purposes - firstly, an examination of how well responsibility models can be understood by domain experts and used to elicit comments and design changes; secondly, an evaluation of the accuracy and utility of responsibility modelling analysis results by domain experts.

The third study aims to investigate the usefulness of formalised responsibility as a constructive technique by requiring non-expert participants to construct responsibility models of a specific domain. Responsibility modelling (like many similar socio-technical modelling techniques) has rarely been applied by modellers other than the creators of the technique; this study examines whether responsibility modelling is an understandable and effective enough technique to be applied successfully without specific experience with the technique.

3.7.2 Case Study Methodology

The research methodology used for the case studies in this thesis is broadly similar to that in previous responsibility modelling approaches such as Sommerville et al. [169]. In their studies, three main techniques are used to elicit information about the problem domain - document analysis, stakeholder interviews and field observations, which each provide different perspectives. Document analysis allows effective elicitation of planned responsibilities (expected behaviour and planned duties), while field observations provide a clearer view of operational responsibilities that may not be formally defined but emerge in response to events. Stakeholder interviews provide a mix of both, indicating how formally defined responsibilities meet with operational reality.

The purpose of these case studies is two-fold: the case studies should demonstrate that responsibility modelling can be effectively applied (in that it can accurately model a range of domains) and demonstrate that it can produce useful analysis and modelling insights for real-world situations. These two outcomes are interlinked - analysis results can only be valid if the model is an accurate (if abstract) representation of the problem system, but the accuracy of a model is itself best tested by considering the model's outputs. This requires a methodology that is able to validate some of the outputs of the modelling process in order to gain confidence that the other, novel outputs of the modelling are valid.

To address this, our methodology requires the use of different information sources; models are constructed based on one category of domain information and are then validated using a different source of information. There are no particular constraints on what types of information can be used at each stage, but we generally begin by using definition documents (manuals, guides, specifications etc.). These types of documents have the advantages of being generally easy to obtain (often being in the public domain), containing substantial detail and describing the 'by the rules' operation of the system.

Having acquired suitable documents we begin by constructing a responsibility model representing the systems described in the documents. It is important that the model is constructed as much as possible based on the selected source documents, rather than using other resources or relying on the modeller's own background knowledge. This aims to limit the subjectivity of the modelling process, improving reproducibility and ensuring that analysis results can be linked back to the original sources.

Having constructed an initial model we then move to validate it by using other sources of information. A wide range of sources can be used depending on the particular domain - stakeholder interviews, field observations, incident reports and academic studies are all possible choices for validation. The use of data generation methods at this stage provides method triangulation [134], which increases confidence and validity in the results of the study. Validation can be performed either interactively or non-interactively. Interactive validation involving domain experts provides the broadest range of validation options, as it can be used to validate either the model itself (by inspecting and discussing it) or the analysis results from the model (by discussing the accuracy and relevance of the results). This tests the model's face validity (according to Sargent [154]) and may provide useful comments and critiques for further revisions to the model.

Non-interactive validation is generally limited to comparing the results of model analysis with analysis results and reported behaviour from other sources, although it may be possible to compare the model with organisational charts or other types of model. This corresponds to the third step of Naylor et al. [132]'s validation process, by testing the model's output against previous occurrences, as well as Sargent's notion of historical data collection.

Validation should be used to gain confidence in the model, and to make any necessary changes in light of the extra evidence. Analysis of the model should be performed at each stage of the case study, firstly using the initial model derived from document analysis and then using each revised model. If the model is valid, the concerns, vulnerabilities and incidents reported in secondary sources should match those raised from analysing the model. This provides confidence and suggests that non-replicated analysis results are also valid. If the results of analysis do not match those in the secondary sources but are not contradicted by them then the situation is unclear, as they may represent issues not previously detected, or they might simply be inaccurate results created by an unrepresentative model. Contradictions between modelling results and the actual system may represent the gap between the 'on paper' and real-world definitions of the system rather than flaws in the modelling process. However, consistent contradictions between modelling results and reported issues may mean that the model has not correctly captured the behaviour of the subject.

This form of case study approach does not necessarily tend towards a clear moment of conclusion. Multiple rounds of model revision and validation may be performed, leading to an iterative cycle of increasing confidence in the validity of the model. This cycle may be brought to an end due to diminishing returns at each iteration; alternatively, it might be halted by a lack of new sources to validate it. The experience of performing the modelling and evaluation process can be as valuable as any concrete end results (a finished model, results of analysis etc.) and the final write-up of the study should provide a clear narrative of the learning points for both the modelling technique and the domain being modelled.

This methodology can broadly be classified as design science or design and creation, although there are some significant differences. The initial stages of design and creation are of considerably lesser importance in our methodology; the awareness and suggestions stages are reduced essentially to the decision as to whether or not the technique is a suitable fit for the specific problem domain and that there is potentially some interesting behaviour to be studied. The development and evaluation stages are more tightly linked; in the design and creation framework the five steps are not necessarily carried out in order and are often iterative - in our methodology the development and evaluation must be carried out in that order, although the pairing is highly iterative. Compared to classical computing science this technique puts greater emphasis on the use of evaluation methods that correspond to real-world behaviour, and encourages the validation of the model using a range of techniques.

Although our methodology is not a direct application of action research it also takes several strong inspirations from that framework. Our methodology does not adopt a strong separation between model construction and model analysis, as both feed strongly into each other. Model construction itself often provides insight into the target system, while analysis detects uncertainties and ambiguities in the model that should be quickly corrected. Like action research, our methodology produces research outcomes at each stage of the modelling and analysis

process, rather than generating only final results at the last step. Likewise, the iterative modelling and analysis steps enables a process of continuous improvement where research can continue as long as new results are achieved; this iterative approach and flexibility is fundamental to action research.

The methodology described here is used to validate individual responsibility models of specific domains. However, responsibility modelling itself is also a (very abstract) model, representing a generic system of responsibilities, actors and resources. Certain aspects of responsibility modelling can be considered as model outputs that should be validated, such as the warnings generated by the use of analysis techniques. These results underpin a core part of the responsibility modelling methodology, and it is important that sufficient confidence in their relevance can be obtained. In the context of design science the responsibility modelling methodology, notation and analysis techniques can all be treated as research artifacts, and should therefore be subject to a structured process of evaluation.

However, we do not seek to validate responsibility modelling in general using a similar technique to this methodology. Firstly, the usefulness and applicability of responsibility modelling in general has already been argued and demonstrated by previous researchers, from which our formalisation directly follows. Secondly, attempting to apply a structured validation approach to such an abstract system model is impractical, as it is not possible to observe a real-world responsibility structure without also capturing abstraction-breaking detail; other validation techniques such as comparisons to other modelling methods are more practical. Instead, performing structured validation of individual responsibility models has the indirect effect of increasing (or decreasing) confidence in responsibility modelling as a whole, and in particular its ability to be applied in particular domain areas. We adopt this strategy in addition to other potential approaches to avoid some of the classic problems arising from 'proof by demonstration' evaluations that attempt to directly evaluate design methodologies but are rendered invalid by failing to address the true scale of real-world problems or to sufficiently involve actual practitioners or domain experts.

3.7.3 Threats to Validity

This type of methodology is subjective, and naturally induces threats to validity. During an iterative modelling process it is inevitable that shortcomings in the model will be detected and corrected. However, if handled inappropriately this can lead to flaws essentially being hidden, giving a false appearance of the final model as error-free. This should be mitigated against by documenting the changes made during each cycle of modelling and analysis, and by publishing not just the final model, but the working versions used at each stage. Conversely, a study that does not detect any interesting problems may be considered unproductive and uninteresting, as it appears that the modelling and analysis technique has failed to deliver any

results. In these cases, a comprehensive discussion of the modelling process and the multiple stages of elaboration remains an important contribution.

It is also important to distinguish between mistakes in performing the modelling process (such as mis-interpretation of a document section) and inherent flaws in the modelling technique or notation. Model changes to correct mistakes in the light of new evidence are to be expected, but are different from making changes due to the modelling notation's inability to capture a particular type of behaviour. Again, the modeller should be as explicit as possible when describing their changes, and changes made without justification should be treated with scepticism. Socio-technical analysis methods suffer from a general problem that there is a tight coupling between modelling experience and domain experience, and it may not be clear if insights are derived directly from the modelling process or from general knowledge of the domain. This is partly addressed by the methodology's focus on using specific source materials at each stage, which tightens the focus and limits the temptation to build a general model from background knowledge. Deviations from this should carefully controlled, and based on a genuine expressed need, such as an absence highlighted in a stakeholder interview.

3.8 Conclusion

Numerous techniques and notations for modelling socio-technical have been developed, and the literature contains a rich range of works based on different conceptual backgrounds and different intended problem domains and modelling outcomes. However, validation of such models remains an important problem. The difficulty of testing models that simulate or predict complex, uncontrollable or unreproducible behaviour has been acknowledged for more than fifty years. From first principles it is argued that validation techniques should attempt to increase confidence in the accuracy and usefulness of models for their specific intended purposes; the metrics used to judge the quality of a model follow from that purpose, although the overall validation methodology should be clearly defined.

A survey of modelling and validation methodologies in socio-technical modelling, requirements engineering and safety analysis has revealed a lack of any consistent application of a formally defined validation methodology, while many publications make no use of validation techniques at all. However, many of those that do implicitly use the same informal validation approach - demonstration of the technique by the creation of a model of a particular system or problem domain, sometimes augmented by comparative analysis of the model. This represents an unfortunate gap in the rigour of socio-technical modelling - while the 'demonstration by case study' validation method has many positive qualities it should not be applied without a suitable understanding of its strengths and weaknesses.

Based on two common research frameworks (action research and design science) and utilising

some of the most common data sources in socio-technical research (interviews, case studies and document analysis) we propose and define a methodologically sound process for evaluating socio-technical models. This methodology focuses on iteratively increasing the confidence in models by continuously analysing and revising models based on the acquisition of new sources of domain information. The acquisition of new information is carefully structured to enable both model evaluation and model development to be conducted using the same information; the modeller must explicitly discuss which parts of the model are supported by the information and which require modifications. Accuracy gaps between the model and reality are a natural part of any complex model, especially in early stages or when working with data sources of limited accuracy, so openness and explicitness in the methodology forces careful discussion of whether these shortcomings are flaws in the model or flaws in the information. As an iterative process the model should tend towards increasing fidelity as more sources of information are obtained - if the reverse is true then it is clear that either the modeller or the modelling technique is failing, and the resulting model is not valid.

We apply this methodology in our case studies of formalised responsibility modelling in Chapters 6 and 7. We do not attempt to define a methodology for validating modelling techniques themselves (such as responsibility modelling), which is a more abstract problem than validating a specific instance model. Instead, we seek to show the general validity of formalised responsibility modelling by demonstrating that it can produce valid and useful models across a range of problem domains.

Chapter 4

Notation & Semantics

4.1 Introduction

Previous versions of responsibility modelling differed greatly in their notation and semantics, both in terms of visual and textual representation as well as in the selection of entities and relationships available. The technique was still in its early stages of development, with individual authors seeking variants more suited to their interests and applications.

These variations make applying responsibility modelling in a consistent way difficult, as no two papers present a unified methodology or notation. Additionally, certain difficult issues of semantics are overlooked or set aside, such as the complexities of delegating responsibilities or the behaviour of relationships when models are transformed. As well as limiting reproducibility and consistency, these shortcomings make providing meaningful tool support and automated analysis difficult.

Some of these issues can be addressed by increasing rigour and standardising the existing notation, or by small changes and expansions to existing concepts. However, some ambiguities in existing responsibility modelling approaches require the introduction of new features to the technique in order to allow precise definitions of complex behaviour and provide the associated benefits of more formal analysis.

While formalised, responsibility modelling retains its flexibility across different system domains and levels of abstraction. Responsibility models are models of the structure of a socio-technical system; in particular, the structure of inter-related responsibilities that should be discharged. We define a responsibility model as a set of responsibilities, resources and actors linked by relations to form a system or systems that aim to discharge a set of responsibilities. As structural models, they do not include specific details about individual responsibilities, such as the physical details of implementation or precise details of their timing. Instead, they show how responsibilities are linked to actors, resources and other

responsibilities. Formalised responsibility modelling is not intended to be a replacement for low-level analysis or domain-specific modelling techniques, as it not designed to capture details below the responsibility level; it is instead intended to model and analyse large-scale socio-technical systems in order to understand the implications of interactions and dependencies between responsibilities.

This chapter will present a well-defined semantics, together with consistent graphical and textual notations for responsibility modelling. We define the semantics using the Eclipse Modelling Framework's Meta-Object Facility [50]. We begin by formalising definitions and representations for the core set of primitive entities and relationships. Differences between the formalisation presented here and other responsibility modelling approaches such as Lock et al. [116] and Sommerville et al. [170] are also discussed.

The next step is the introduction of a constraint or specification language, which allows for the precise description of complex relationships. A significant advance is provided by the introduction of explicit uncertainty - a notation for capturing uncertainty (lack of domain knowledge or inherent randomness) allows modellers and analysts to differentiate between unspecified areas due to simplification or scope reduction and areas where full specification is not possible.

Formalised responsibility modelling incorporates mechanisms for assessing the timeliness of responsibility discharge and techniques for modelling multi-layered domains as well as defining possible model transformations. Finally, two examples demonstrate the use of both basic and advanced modelling features in socio-technical domains.

[An earlier version of the initial parts of this chapter was published as:

Robbie Simpson and Tim Storer. Formalising Responsibility Modelling for Automatic Analysis. *Lecture Notes in Business Information Processing*, 231:125–140, 2015. doi: 10.1007/978-3-319-24626-010]

4.2 Meta-Model

The structure of responsibility modelling is formally defined using the Eclipse Modelling Framework's [50] Ecore meta-modelling system, as shown in Figure 4.1. Ecore is an implementation of the Object Management Group's EMOF (Essential Meta-Object Facility) standard [135], which provides a domain-specific language for defining meta-models, producing a type system and rules for interactions between objects. Ecore is used to produce a definition of responsibility modelling's semantics and to generate a skeleton of Java classes representing the meta-model, which aids the development of tool support.

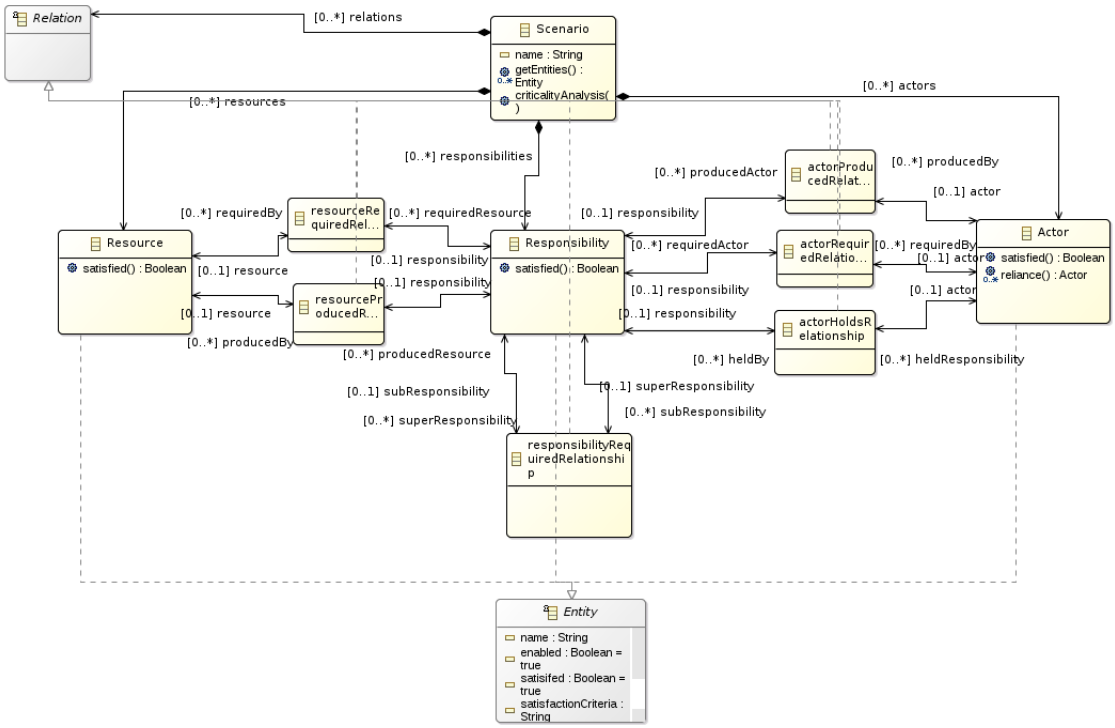


Figure 4.1: Ecore / EMOF Meta-model for responsibility modelling

The different objects within responsibility modelling (responsibilities, actors and resources) are represented in the meta-model as entities. Properties common to all are defined in a generic entity type, while responsibilities, actors and resources are each implemented as specific types extended from the original entity type. All the standard entity types (Resource, Actor, Responsibility) are contained within a higher-level Scenario entity.

Similarly, relations are first defined with a general relations type, while each particular variant of relation is defined as a specific class in one-to-many associations with the appropriate entities. This enforces constraints on the possible relations between different types of entity; for example, it is impossible for an actor to be part of a production relation. Relations are always bi-directional - each entity has full knowledge of all the relations it forms part of.

4.3 Primitive Entities

The core of the responsibility modelling language is a set of three primitive entities: responsibilities, actors and resources. Previous versions of responsibility modelling have included special subdivisions of these three entity types. For example, Lock et al. [116] differentiates between physical resources and information resources and between human agents and organisation agents; other papers distinguish human actors and automated actors. Other entities have also been introduced - Sommerville [166] refines responsibilities into goals that are satisfied

by following processes.

We have chosen to use the smallest possible set of entities in our formulation of responsibility modelling. Limiting the number of core primitives should increase the ease of understanding the technique and greatly simplifies the operation of automated analysis and tool-supported modelling. Removing the specialised entity types may shift some effort from the modelling phase to the analysis phase, as there are valid reasons to treat different types of actors or resources differently, such as distinguishing between human actors and machine actors when considering load or dependency. However, these distinctions are not absolute, and some level of extra consideration when analysing would always be required. Assumptions made about the types of entities may not always hold; for example, information resources may not always be easily transferable (consider heavy boxes of files) and human actors may be instructed to only follow very tightly defined routines.

Choosing to remove the additional, separate entity types (goals and processes) has only limited effects on the expressiveness of the technique, as they can often be represented using just responsibilities. Many goals can be simply rephrased as responsibilities, and drawing a clear distinction between goals and responsibilities is difficult - an abstract responsibility such as a policeman 'upholding the law' is clearly a goal, but is also definitely a responsibility. Processes are a less natural fit for responsibilities, as the concept of responsibilities applies less clearly to small, well-defined tasks. However, the notation remains suited to modelling them; an approach that responsibility modelling papers after Sommerville [166] took, despite the reduction in intuitiveness.

Entities can be represented in either a graphical or textual form. Figure 4.2 provides a simple example of the graphical notation for entities, produced using a responsibility modelling toolkit. Figure 4.3 shows the textual representation. The graphical model also uses the basic relationship types, which are described in detail in Section 4.4; these relationships have been removed from the textual model for clarity.

4.3.1 Responsibilities

Responsibilities are the core concept in a responsibility model of a socio-technical system. The modelling language in this thesis adopts the definition of a responsibility provided by Sommerville et al. [169], as an abstraction to describe behaviours in a socio-technical system as a duty or obligation that should be discharged by actors in that system. Therefore in responsibility modelling the behaviour of a system is characterised by the discharge of responsibilities (both successful and unsuccessful) by system actors.

Responsibilities can represent behaviour at a range of levels of abstraction, from high level duties (example: maintain law and order) through to specific implementation strategies

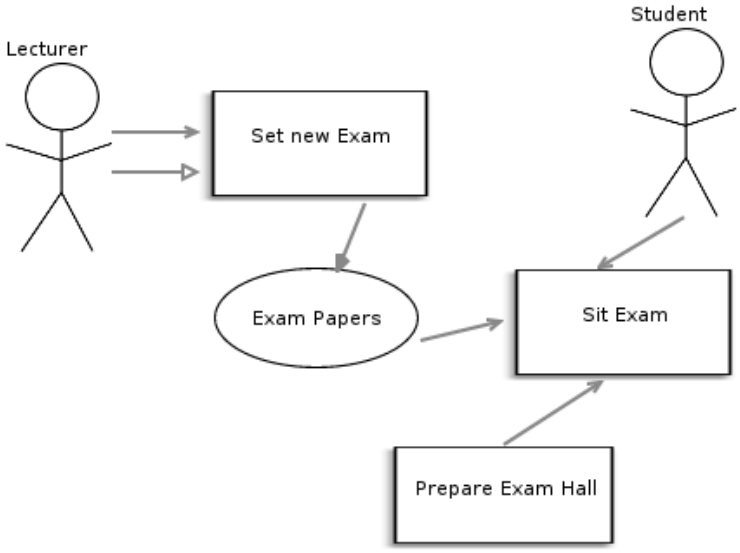


Figure 4.2: Responsibility model for a university exam: The Exam Papers resource is produced and consumed; the Sit Exam responsibility is dependent on both the actor Student and the responsibility Prepare Exam Hall. The Prepare Exam Paper responsibility both requires the Lecturer and is held by them.

```
Sit Exam::  Responsibility
Set new Exam::  Responsibility
Prepare Exam Hall::  Responsibility

Exam Papers::  Resource

Student::  Actor
Lecturer::  Actor
```

Figure 4.3: Textual responsibility model for entities in university exam

(example: complete charge sheet). However, there is a separation of concern between the responsibilities that must be discharged (and the conditions under which a responsibility is considered to be discharged) and the manner in which the discharge takes place.

Graphically, responsibilities are represented by simple rectangles with the name written inside. Textually, they are represented in the form NAME:: Responsibility.

4.3.2 Actors

Actors are entities within the system that are able to discharge responsibilities. Actors in responsibility modelling do not necessarily represent individual people in the real world; actors can flexibly represent roles defined within the system. These roles may represent individual positions ('Head of Security'), computational or technical systems ('Payment Processing Software', 'Mechanical Deadlocks') or organisations ('Police Scotland').

In many modelling cases it is not necessary to know which actual individuals fit these roles; it is often enough to simply know the role is filled. However, there is nothing stopping one individual or organisation filling multiple roles in the same system, which can lead to vulnerabilities. As a result, it can be necessary to analyse or constrain the allocation of roles; options for doing so are discussed later in Chapter 5.

Graphically, actors are represented as stick-figure characters with their name attached. Textually, they are represented in the form NAME:: Actor.

4.3.3 Resources

Resources represent any objects that are produced or consumed by or within the socio-technical system. Resources can be physical items or digital information; they can also represent more abstract concepts such as authorisations and goodwill.

Resources are either available or unavailable, depending on the current state of the system. They do not have quantities; either a sufficient amount of the resource exists or it does not. Likewise, the quality of resources is not specified - the resource is either of sufficient quality and sufficient size (and so is available) or of insufficient quality or size (and hence unavailable). This simplifies modelling by removing the need to specify the output levels or productivity of processes.

Graphically, resources are represented as rounded ovals with their name written inside. Textually, they are represented in the form NAME:: Resource.

```
Lecturer => Set new Exam
Lecturer -> Set new Exam

Set new Exam -> Exam Papers

Exam Papers -> Sit Exam

Student -> Sit Exam

Prepare Exam Hall -> Sit Exam
```

Figure 4.4: Textual responsibility model showing relationships, using the entities defined in Figure 4.2

4.4 Relations

The core entity types can be linked using five kinds of relationships. Resources can be *produced* or *consumed* by responsibilities; responsibilities can *require the discharge of other responsibilities* or *require certain actors*; actors can *hold* responsibilities.

These five relationship types broadly correspond to those used in previous versions of responsibility modelling. The main difference is the introduction of the 'holds' relationship between actors and responsibilities; previously, a 'responsible for' relationship was used that combines elements of both 'holds' and 'requiredActor'. This distinction emphasises the different ways an actor can be linked to a responsibility; they can be required on a practical level to complete the responsibility in some way ('requiredActor') or hold some authority or accountability over it ('holds'). These two variations are not mutually exclusive, and are discussed further in Section 4.4.2.

Figures 4.2 & 4.4 show the graphical and textual representations of relationships, respectively. Graphically, all relationships share the same basic arrow representation, with the exact relationship type being determined by the two entities linked and the direction of the arrow. For example, a link between a resource and a responsibility leading from the resource to the responsibility is a consumption relationship, while a link in the opposite direction is a production relationship. The sole exception is the difference between the 'holds' and 'required actor' relationships; these both share the same direction, so 'holds' is indicated by the use of an unfilled arrow head. Textually, all relationships use the same dash and right angle bracket notation (->) with the exception of holds, which uses an equals and right angle bracket (=>).

4.4.1 Production & Consumption

Resources are produced or consumed by responsibilities; a responsibility requiring resources cannot be successfully discharged if the required resources are not available. Resources cannot be directly created by actors, as the responsibility reflects the duty, obligation or intent behind the resource creation activity.

Resources are successfully produced if a responsibility that produces them is successfully discharged. Additionally, resources without any producing responsibilities are by default available. When produced, resources are in effect available immediately and there is no concept of waiting for production to complete; this is a consequence of responsibility modelling's temporal model, which is covered in Section 4.8.

Resource consumption by responsibilities indicates that those resources is required to complete the responsibility. The alternative description 'required resource' offers a clearer description of the underlying behaviour, as resources that are consumed by one responsibility are still available for consumption by other responsibilities; as available resources are always considered to exist in sufficient quantities it is not possible to use up the entire capacity of a resource.

4.4.2 Actor Assignment

Relationships between actors and responsibilities can take two distinct forms, although in many cases both forms apply. Firstly, actors can be required for a responsibility to be discharged - the activity of the actor is in some way necessary for the responsibility to be met. Secondly, an actor can be accountable or 'responsible' for a responsibility, but not be required to play an active role in its discharge. The first relationship is formulated as the 'requiredActor' relationship, while the second is represented by the 'holds' relationship. This is akin to the notion of the cause & answerability / accountability definitions of responsibility by Cholvy et al. [29].

Where a responsibility is linked to an actor by a 'requiredActor' relationship it is necessary for the actor to be active in order for the responsibility to be discharged; if the actor is disabled in any circumstance it is not possible to complete the responsibility.

When an actor is linked to a responsibility with a 'holds' relationship (and is not also linked by 'required actor') the actor is not necessary for the completion of the responsibility - if the actor is disabled the responsibility continues as before. However, if this leaves the responsibility not being held by any actors then there is a risk of organisational confusion, as the responsibility is required to occur without any particular actor being accountable for a failure to discharge it.

These two relationships are closely related, and in the majority of cases both types of relationship will exist. Most versions of responsibility modelling use the assignment of actors to responsibilities in the sense of the definition given in Lock et al. [115] : 'A duty, held by some agent, to achieve, maintain or avoid some given state... It also encompasses aspects of accountability'. Modelling in this style can easily be achieved by always using both forms of relationship together and only using them individually in special cases.

This separation of the active role and the accountability role is not new. Sommerville et al. [170] use a notation where delegation of responsibilities transfers the active element (which they term 'responsible for') to a new actor, while the original (or creating) actor remains an 'authority' for the responsibility (who decides if the responsibility has been correctly discharged, and is some way accountable for the result).

The ability to separate the two relationships becomes most useful when considering fairly abstract responsibilities. For example, in the British government each department is headed by a senior minister, who is the public face of the department is accountable to the public and her party colleagues. The actual operations of the department are carried out by civil servants, the most senior of which is the Permanent Secretary. In practice this means the minister 'holds' the responsibilities of the department (as they take the credit or blame, but the department can function perfectly well without them) while the Permanent Secretary is a 'requiredActor' (without them the management of the department is severely impaired, but they hold no public responsibility for the department's actions).

4.4.3 Required Responsibilities

Responsibilities can also require other responsibilities to be successfully discharged; the requiring responsibility can only be discharged if the required responsibility is also discharged. There is natural overlap between this 'required responsibility' relationship and the more general pattern of a responsibility producing a resource, which is then required by another responsibility. The resource creation and consumption pattern is best suited to cases where the resource is meaningful, such as when it can be produced in multiple ways or when its presence is audited by another responsibility. In other cases the 'required responsibility' relationship is more suited, as it does not require the creation of potentially unnecessary extra resources. For example, modelling with a resource is suited to situations where a responsibility produces a notification of success (an alert, a report etc.), while directly using a required responsibility relation is suited to situations where the success of a previous responsibility must be taken on trust, without notification.

The notion of a required responsibility is similar to but distinct from the previous concept of sub-responsibility. In Storer and Lock [175], decomposition is formally defined as a transformation on an existing responsibility that replaces it with a set of sub-responsibilities which

remain bound by the same relationships as the original responsibility. Later, Sommerville et al. [169] features decomposition where the original responsibility remains intact; this represents refinement and composition rather than full transformation and replacement. The first meaning of sub-responsibilities is really a model transformation, which will be defined in Section 4.10.

The notion of the 'required responsibility' is closer to the second meaning, but some differences remain. In classic responsibility modelling, a sub-responsibility relationship indicates that the sub-responsibility is required to complete the original responsibility, and this is true of 'required responsibility'. However, it also indicates that the sub-part is a component or integral part of the original responsibility; in effect, it is a element of the original responsibility that has been chosen to specify in detail rather than abstract away, usually because it features some interesting behaviour or requires specific actors or resources. This provides very little extra expressiveness (as the meaning of a sub-responsibility varies between different contexts, and so little can be inferred) while not allowing the direct modelling of responsibilities that are interrelated but not sub-parts of each other.

This is not true of the 'required responsibility' relationship defined here, which merely indicates the need for the responsibility without implying that it is part of the original responsibility. For example, the responsibility of a voter to 'Vote in election' requires that candidates discharge the responsibility 'Stand for election' so that there are candidates to vote for; however, standing for election is clearly not a sub-unit of voting in an election. This provides extra flexibility in specifying interrelated responsibility, while the expression of explicit decomposition is discussed in Section 4.10.

4.5 Delegation, Supervision & Dependency

In many socio-technical systems responsibilities are initially held by an actor that then delegates away some or all of these responsibilities. This often involves the refinement and decomposition of high-level responsibilities - the original actor holds the top responsibility, and the various lower-level responsibilities that contribute to it are held by different actors.

In previous versions of responsibility modelling this would be modelled using sub-responsibilities. By default, sub-responsibilities would be assigned to the same actor as the main responsibility. If a sub-responsibility is explicitly assigned to a different actor, then that responsibility has been delegated. A different approach is adopted in this formalisation of responsibility modelling, as requiring a responsibility does not necessarily imply that that responsibility composes part of another. If explicit decomposition is desired, a model transformation is performed instead (see Section 4.10).

As a result decompositions of responsibilities are not declared explicitly in formalised responsibility models. Rather than modelling 'as defined' hierarchical responsibility and actor relationships it instead models 'in practice' structures. Previously, implicit supervisory relationships could be identified by locating sub-responsibilities that had been delegated - the discharge of the original responsibility requires the discharge of the sub-responsibilities, and so the actor holding the original responsibility is dependent on the success of the actors holding the sub-responsibilities. The use of sub-responsibilities implies a hierarchical organisation structure within the system, and so this could be considered an example of senior actors supervising junior actors. In many socio-technical systems this can enable useful analysis, as the hierarchical model generated by considering sub-responsibilities may well differ from the formal managerial structure, for example.

However, this approach can be limiting, especially in multi-organisation systems that embody Systems-Of-Systems like behaviour [114]. In this context the hierarchical concept of supervision is inappropriate - the relationships between actors are not hierarchical, and instead reflect collaborative or contract-based agreements. The underlying meaning of the relationship is the same - an actor can only complete their objectives with the support of other actors, so there are clear benefits to some form of oversight or supervision. We call this implication 'dependency', which is effectively a generalisation of the original 'supervision' concept.

Dependency occurs in any case where an actor directly or indirectly relies upon another actor. Dependency can occur directly via actors (an actor holds a responsibility that requires another responsibility to be successfully discharged) or indirectly via resources or responsibilities (an actor holds a responsibility that requires a resource or responsibility, and this other entity requires a different actor to be discharged). Dependency relations can be very deep and difficult to detect by simple inspection - the actor depended on may exist at the far end of a long chain of resource and responsibilities.

4.6 Constraint Language

Entity relationships provide the basic building blocks for constructing responsibility models, and can be used to express complex interdependencies such as delegation. However, the exact semantics of relationships can become unclear and limiting in more complex cases.

For example, it may be necessary to model a situation where two actors are in an Required Actor relationship with a responsibility. Does this mean that both actors are necessary to discharge the responsibility, or that either actor can do it alone? Both of these are perfectly plausible scenarios, but using just relationships alone cannot distinguish between the two different cases. We might also wish to indicate that one actor is the default and that the second

actor only intervenes if the first is unavailable; again, this cannot be clearly expressed using only sets of relationships.

Other scenarios involving multiple interacting entities are also impractical to model. Many systems feature elements where responsibilities can be discharged if a certain threshold of entities are successful - for example, a voting circuit such as used in safety-critical systems. A small-scale threshold problem can be modelled using the existing notation by pairing together individual elements as sub-responsibilities, but this quickly leads to a complex model as the number of entities increases.

Fundamentally, the additive nature of conventional relationships helps comprehension and ease of modelling, but does not allow the specification of many common and significant parts of socio-technical systems. We therefore propose a *constraint language* of responsibility which allows the specification of *satisfaction criteria*; complex relationship details and pre-requisites for entities can be specified in a predicate logic. By default, all relationships maintain their classical meanings and priorities, but exact definitions can be given for shared responsibilities and alternative discharge options where required.

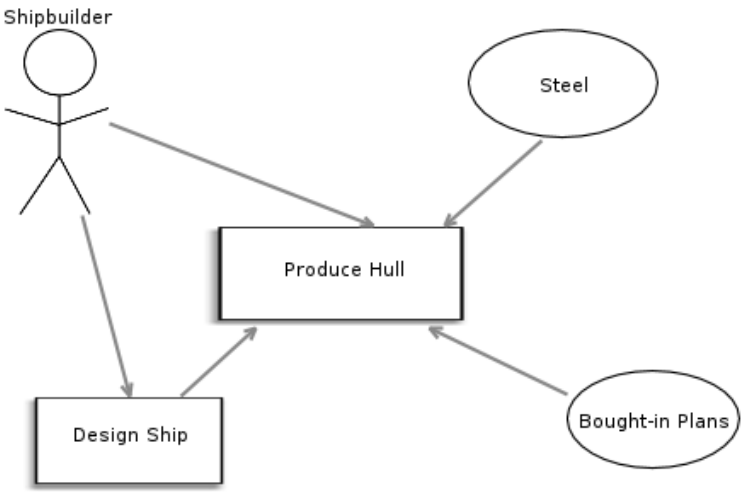
A responsibility is discharged or a resource produced if all related entities are available; for example, a responsibility is successfully discharged if all required actors are active, all required resources available and all required responsibilities discharged (a logical conjunction of all required entities). However, the constraint logic allows for specification of more complex scenarios. All types of relation can be augmented with constraint logic, with the exception of 'holds', which represents accountability and authority rather than the functional requirements of the other types.

Figure 4.5 shows an example of a model defined using satisfaction criteria. In this example, a shipbuilder can produce a vessel either by designing the ship themselves, or by buying in pre-existing plans. The resource and actor requirements take their default condition of conjunction, as expressed by the & operator. Specifying the either-or nature of the different design requires the use of the || operator to distinguish the OR relation, and the use of parenthesis to limit the operator to the appropriate items.

More generally, any combination of AND and OR relationships can be specified and bracketed using parenthesis to any necessary depth. The criteria are evaluated left-to-right and top-down, subject to bracketing. This allows for the modelling of complex combinatorial relationships, including threshold cases like voting circuits.

When no satisfaction criteria are explicitly defined, a conjunction of all the defined relationships is assumed. For example, in the ship building case without the given satisfaction criteria for Produce Hull, the diagram would be interpreted as a conjunction of all the related entities:

Produce Hull:: Shipbuilder Active & Steel Exists & DesignShip Discharged & Bought-in Plans Exists



Produce Hull:: Shipbuilder Active & Steel Exists & (DesignShip Discharged || Bought-in Plans Exists)

Figure 4.5: Example responsibility model showing constraint logic satisfaction criteria

This form of constraints allows specification of different options for satisfying responsibilities. However, it does not specify the ordering or likeliness of a particular technique being used; it can specify a choice of two options, but not distinguish between the 'normal' option and the 'emergency' option. This limits the precision of analysis techniques based on assessing the number of relationships an entity is involved in - an actor required in many fall-back cases (such as the emergency services) may appear to have a crushing load, when in reality it is highly unlikely that they will be required by all responsibilities at once. Assessing the maximum possible load on an entity is still a useful technique, but it is also beneficial to capture scenarios where there is a sudden increase in demand.

This is supported by extending the constraint logic to incorporate ranked priorities. For example, the shipbuilding example can be amended to specify that designing the ship in-house is the preferable option, and that plans will only be bought in if that responsibility cannot be discharged:

Produce Hull:: Shipbuilder Active & Steel Exists & ([1] DesignShip Discharged || [2] Bought-in Plans Exists)

The [n] notation indicates that the different discharge options have a specific order, with lower-numbered options being preferable to higher-numbered options. This replaces the standard 'joint and equal' format of requirements, and so allows for more nuanced evaluations of the load on different entities. Entities that are required as a second or third priority option clearly receive less load than those required as a first option, which can be reflected in analysis

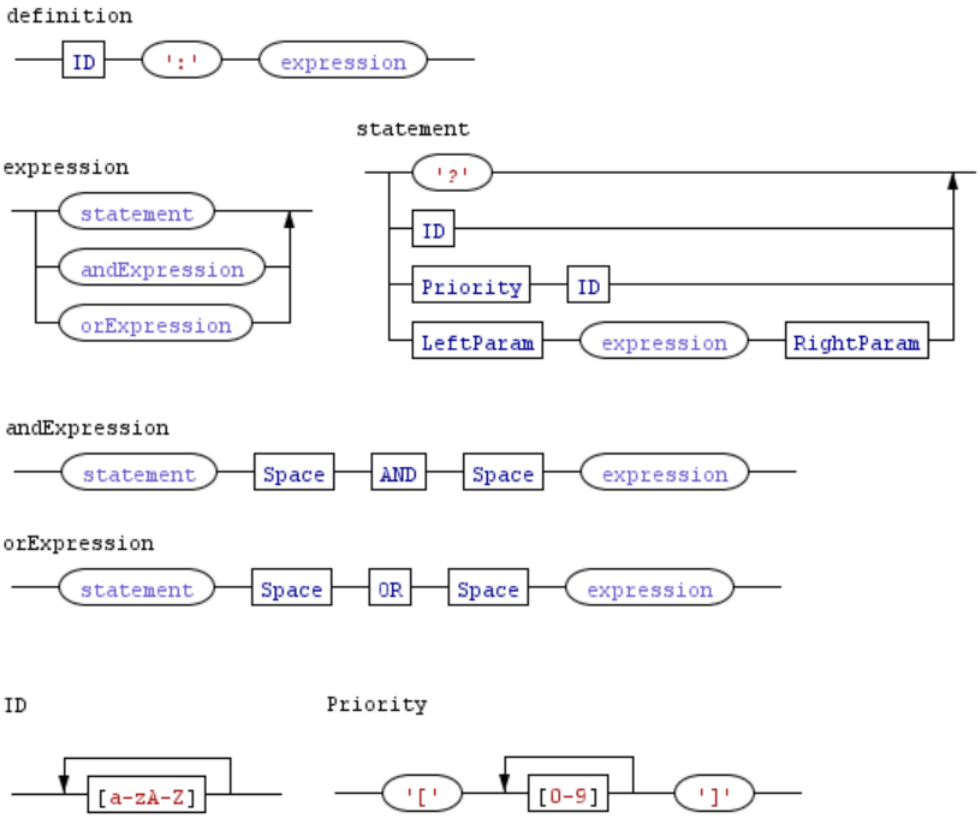


Figure 4.6: Syntax diagram for the constraint language

techniques. The exact treatment may vary depending on the analysis technique and context; in some cases it may be sensible to discount the load from low priority relationships entirely, while in other cases weighting down the low priority relationships on a sliding scale may be more appropriate.

The syntax diagram in Figure 4.6 provides a formal representation of the constraint language grammar, incorporating explicit uncertainty as described in the next section.

4.7 '?' - Explicit Uncertainty

Most socio-technical modelling techniques make the implicit assumption that their models are a complete representation of a problem domain - details may be abstracted, but all possible scenarios can be covered by them. This assumption requires the modeller to have a perfect understanding of the domain, which is impractical in the vast majority of cases. Such a strong assumption can be supported when modelling very tightly defined problems (for example, interlocks in safety-critical systems) but not in the wider body of socio-technical systems. This leads to a variety of potential shortcomings in modelling. In order to produce a fully-defined

model the modeller may focus on very specific parts of a system that can be effectively captured, which can neglect the overall socio-technical context of the system. Alternatively, complex details may be overlooked in modelling in favour of producing a complete model, but this missing detail weakens the conclusion of any analysis. Finally, attempting to produce a model that is both fully detailed and that covers the full scope of the problem is likely to result in an intractably complex model.

Generally, any model of a system will feature two types of uncertainty [100]. Epistemic uncertainty is uncertainty 'about the world' - uncertainty caused because the modeller does not have enough knowledge about the problem to specify it exactly. This type of uncertainty can (in theory) be eliminated or greatly reduced by acquiring more information about the domain - more precise measurements, longer observations, more source materials and so on.

Epistemic uncertainty may also occur when a modeller deliberately limits the information they use when constructing a model - for example choosing to ignore edge cases in order to produce a simpler but less accurate model. A similar uncertainty is introduced when models are refined to produce sub-components. Sometimes elements are refined completely, such that the element is composed entirely of the related sub-parts. In other cases, the sub-parts reflect only part of the original element - usually elements that are of special interest to the modeller. In the first case it is possible to determine the validity of the original element by considering the subelements; in the second case there are factors influencing completion that have not been explicitly modelled. As a result, not performing complete refinement introduces epistemic uncertainty by leaving some of the model behaviour undefined.

The second form of uncertainty is aleatory uncertainty - uncertainty 'in the world'. This arises where there is an inherent randomness in the system being modelled that cannot be eliminated by defining the system in more detail. This can occur because of fundamental physical or social reasons (no system can exactly measure both the position and velocity of an object) or because the information required is outside the agreed scope of the system. In particular, aleatory uncertainty can occur in socio-technical modelling where different viewpoints of the system (actor interviews, system documents, ethnographic studies etc.) give inconsistent results; this indicates that there is uncertainty within the system itself, which clearly cannot be resolved by more measurement. The inconsistency leads to unpredictability in the system operation.

These two forms of uncertainty arise from different causes, but their effects on the modelling and analysis of socio-technical systems are very similar. The difference between the two types has been long discussed, and it has been argued that they are essentially the same [100]. Both introduce a level of imprecision to system models, and hence limit the confidence and accuracy of analysis implications derived from those models. Once a model is constructed, the two types of uncertainty are effectively indistinguishable, as they both produce the same

results. The difference occurs primarily at the modelling phase - epistemic uncertainty can be minimised by more extensive study of the domain, but aleatoric uncertainty is irresolvable.

Explicit handling of uncertainty is rare in socio-technical modelling techniques, despite socio-technical systems often containing high levels of uncertain behaviour. A notable exception to this is the 'SeeMee' modelling method [75] which allows for the explicit expression of 'vagueness' within models. This notation enables explicit statement of incompleteness for abstraction purposes as well as due to a lack of detail about the system or a lack of confidence about the modeller's understanding of the system, and can be applied both to the definitions of entities and to relations between those entities. However, vagueness is discussed only in terms of the modeller's understanding of the world (epistemic uncertainty) and SeeMee does not appear to support explicit expression of aleatory uncertainty.

Both forms of uncertainty can be explicitly expressed in responsibility models by extending the constraint logic. By default, any set of satisfaction criteria completely define the corresponding responsibility, and there is no uncertainty or randomness explicitly modelled. (However, this does not exclude the possibility of the model being inaccurate due to errors in the modelling process) If a responsibility does feature uncertainty (either epistemic or aleatoric) this can be explicitly specified by including the symbol <?> in the satisfaction criteria for the responsibility incorporating the uncertainty. For example:

Protect Military Base:: SecureGates Discharged & CheckForTunnels Discharged & AirSpaceSecured Discharged & <?>

states that the modeller is aware there are other failure cases than the three dependent responsibilities listed, but is either unable or unwilling to explicitly model all of them. This is primarily aleatory uncertainty - more elaborate modelling can include more forms of safety checks, but providing exhaustive security is ultimately impossible.

In contrast, consider attempting to model counter-terrorism activity in the UK:

Intercept Terrorist Communications: GCHQ Active & <?>

It is clear that GCHQ (the UK's signal intelligence division) is involved, but it is very difficult to be any more specific given publicly available information. There are certainly additional requirements needed to satisfy this responsibility in practice (international collaboration, listening equipment, cryptographic backdoors etc.) but it is impossible to specify these accurately. As a result, the best modelling compromise is to specify only the elements that is it possible to be certain about, and to use <?> to indicate the substantial epistemic uncertainty.

Introducing uncertainty to a responsibility also changes the way that the discharge of a responsibility can be evaluated. The presence of uncertainty indicates that the modeller or analyst cannot perfectly predict whether or not the responsibility is discharged; they may be able to determine cases where the responsibility will definitely succeed or definitely fail,

but may not be able to generalise across all occurrences. This introduces the concept of 'conditional' success, where the final result rests solely on the modelled uncertainty.

For example, consider a model of the responsibility for safely storing some important documents:

SafeStorage:: DocumentsUndamaged Discharged & <?>

In this fragment, it is defined that the documents being undamaged is a necessary (but not sufficient) criteria for success. The responsibility will definitely not be discharged if the documents are damaged, but may also not be discharged in some other case (e.g. the documents could be stolen) as encapsulated by the explicit uncertainty. This leaves two possible outcomes for the responsibility - it can definitely fail to be discharged (damaged documents) or achieve 'conditional success' - it will not fail based on the information available to the model, but success is not guaranteed.

Conversely, a responsibility may have defined success criteria but undefined failure criteria:

EmergencySurgery:: PatientAlive Discharged || <?>

In medical surgery on a critically injured patient the responsibility may be considered a success if the patient lives, regardless of any side effects or complications - the severity of the injury means that any survival is a success. However, the death of a patient does not necessarily reflect a failure on behalf of the doctor or health organisation; despite their best efforts some fatalities are inevitable. Of course, it is still possible for the operation to be a failure in a particular case if the patient dies due to avoidable negligence. In this case, the responsibility accurately represents the two possible outcomes - either success (the patient lives) or 'conditional success' (success depends on individual medical details that are not modelled).

This extension allows for clear specification of uncertainty and incompleteness, but only when the modeller is able to acknowledge the existence of this uncertainty. It does not address the 'unknown unknowns' - uncertainties that are not recognised during the model process. The strength of the extension is its ability to capture the 'known unknowns', allowing the modeller to document areas of uncertainty, rather than forcing them to make unjustifiable assumptions of perfect information and predictability.

Incorporating explicit uncertainty into modelling also changes the analysis that can be performed on these models. Without uncertainty the model is assumed to be a complete (subject to some assumptions) representation of the problem domain, and hence assertions and proofs made on the model can be directly transferred to the underlying domain. This is no longer true in responsibility modelling, as it is explicitly acknowledged that the model contains uncertainty. Consequently, responsibility models can be used to identify vulnerabilities in the responsibility structure of the socio-technical system, but cannot be used to assess the

robustness of the system with respect to underlying implementation details.

Based on this, analysis can produce three possible results. If it is possible to identify an undischargable responsibility in any model, then it can be stated that the system may fail due to a responsibility vulnerability. If no undischarged responsibilities exist, and no <?> is present then the system will not fail due to vulnerabilities in the responsibility structure and it can be stated that there is no undefined behaviour that could cause it to fail.

If there are no undischarged responsibilities, but <?> is present then it can be stated that the system will not fail due to the responsibility structure. It is not possible to claim definitively if the system will work due to the undefined behaviour indicated by <?>, but it is safe to say that there will be no vulnerabilities caused by the system structure proposed and modelled. Of course, unmodelled implementation details may cause the system to fail in practice. The advantage of formalised responsibility modelling is to isolate and be explicit about the areas of the model that could represent potential failures.

4.8 Temporal Behaviour

Previous responsibility modelling research has adopted an informal approach to incorporating temporal behaviour in modelling. Responsibilities are discharged, resources are produced and consumed etc. without defining how long it takes for these actions to occur, or the order in which they occur. As a result, it is not possible to incorporate in a responsibility model the concept of 'timeliness' of action - the action will occur at some point, but it is not possible to determine when.

Temporal behaviour is only implied by the domain context - a responsibility to rescue evacuate flooded houses should happen before the inhabitants drown; however, this becomes difficult to interpret when a model features activities that are clearly at different orders of magnitude in duration or response time. Additionally, this complicates the analysis of dependencies between entities. The discharge of a responsibility may rely on a resource, and this resource may be produced by another responsibility. This implies a temporal ordering - firstly the resource production responsibility must be discharged, and only then can the resource consuming responsibility operate. The resource production responsibility has a clear potential to delay the consuming responsibility and hence the overall function of the system, but the temporal model of responsibility modelling lacks a defined semantics to interpret this.

Later work on responsibility modelling recognised the benefits of this form of analysis. HAZOP-style keywords were introduced to the methodology and the temporal keywords Early, Late & Never feature significantly. These keywords are used (such as in Lock et al. [115]) to analyse the consequences and severity of variations in the timeliness of responsibility discharge. The lack of any underlying semantics for timeliness means that the analysis is not

directly performed on the model; the analyst applies the keywords to the list of entities and uses their understanding of the domain to predict the likely impact. A further disadvantage is that it is difficult to assess cascade effects across whole responsibility models rather than just single responsibilities.

To address this, formalised responsibility modelling adopts a single time frame within which responsibilities are discharged. This approach is similar to the concept of 'epoch time' used by the BAN logic [25], where time is divided simply into the past and present. Burrows et al. [25] argue that such an approach simplifies the modelling of problems without losing significant modelling precision. Under this formalism, a resource produced by a responsibility will be produced 'eventually', but it could take an unlimited amount of time. Similarly, a responsibility missing a required resource will 'eventually' fail in some circumstance where the resource is called upon - but the socio-technical system could operate for its entire lifetime without the failure actually occurring. This representation in the responsibility model does not represent direct failures but potential vulnerabilities which may never actually occur in practice.

A useful analogy is to the clock cycle of an integrated circuit. The chip is configured to perform various calculations and determine results by using various intermediate steps implemented with simple logic gates. Different routes through the chip take different times to complete; reading output values before the clock cycle ends may provide inaccurate results. Setting the correct clock cycle guarantees that all calculations will have been completed and results propagated correctly. Responsibility modelling is similar - after some arbitrarily long time period the system will operate as modelled, but expecting results too quickly is risky - sometimes the system will operate exactly as expected, but at other times incomplete activities will not function as intended.

This approach to timing offers a number of useful simplifications. As bounds are not placed on the time needed to discharge responsibilities or produce resources the problem of race conditions is eliminated - required entities are either ready in time or never available at all. It is not necessary to carefully evaluate the discharge speed of entities, instead only judging if the time taken is acceptable or unacceptable. It provides a meaningful semantics for actors holding multiple responsibilities - they will ensure that responsibilities are eventually completed, but this does not mean that they are working on all the responsibilities at once. Similarly, with shared responsibilities there is no need to know which actor(s) are discharging the responsibility - only that it will be eventually discharged.

The analysis methods are unaffected by this paradigm, as they are based on structural rather than temporal principles. Some HAZOPs-type keywords do invoke temporal behaviour, such as Early/Late discharge of responsibilities. In these cases, the timeliness (or not) of responsibility discharge is local to other responsibilities in the system, rather than absolute.

Responsibility modelling has not previously used strict definitions of time, and this relativity has always been implicit when performing this type of analysis.

4.9 Instantiation

Responsibility models are abstract representations of a particular system, even though they are constructed in the context of a specific instance. For example, the model of flood response in Sommerville et al. [170] is based specifically on the city of Carlisle, but the entities used in the model remain generic - 'Fire Brigade' rather than 'Cumbria Fire' or specifying a particular fire station. Likewise, a responsibility is defined as 'Establish Reception Centres', rather than 'Establish City Hall reception centre' or 'Establish reception centre at University of Cumbria'.

This abstraction can be useful in hiding unimportant or irrelevant aspects of implementation - for example, the overall structure of flood response does not depend on the particular order in which streets are evacuated. It is also provides a degree of flexibility in the analysis of different scenarios; for example, operational changes to implementation details may not require corresponding changes to the responsibility model.

However, excessive use of abstraction can hide specific details that may be important. For example, an emergency plan may span an area that has several different police forces or fire brigades; if only the generic actor is used, then it is unclear which force (all of them?; some of them?; just one?) is responsible for assuming the responsibilities of that actor. In particular, there might be complex interactions between linked entities - one police force responsible for urban reception centres and another responsible for rural reception centres, for example.

More generally, this relationship between different levels of abstraction of responsibility structure can be useful for modelling insight and analysis. Socio-technical systems can be defined at multiple levels, and the interactions and mappings between these levels represent a significant amount of the system's complexity (such as Leveson [109]'s concept of a hierarchy of levels). Many important questions about such systems can be answered by considering the links between views of the system at different abstraction levels - for example, whether a proposed government program is compatible with the current law, or whether a fully developed software system actually matches the original requirements.

To enable this the concept of *instantiation* is introduced - the idea that one responsibility model can contain entities at different levels of abstraction. Each level (or layer) of the model can contain the full range of entities and can be linked across levels by realisation relations - uni-directional relationships that indicate that the source entity is a specialisation or concrete example of the target entity. This allows modellers to explicitly indicate implementation details where relevant, without affecting the semantics of the general model. Instantiation allows mappings between all types of entities for both abstract and specific cases.

Currently, these relationships serve a descriptive and indicative function, and do not affect the results of any formal analysis. As a result, they are best used to visualise potential implementation options (such as for discussion with stakeholders) or to assist with manual evaluation and analysis (such as displaying instance-level actors for use with role analysis). Automatic analysis techniques using these relations are certainly possible, but are rendered difficult by ambiguities in parts of the relation semantics and hence are reserved for future work.

4.10 Transformations

As part of the modelling process it is often necessary to refine and expand an existing responsibility model. In many cases modellers may wish to modify one part of an existing model (such as to model a particular aspect in more detail) without deliberately changing the rest of the model. This can be complex when the parts to be modified are linked by relations to other entities, potentially leading to unintended effects on other parts of the model. The need for a consistent way to perform these changes was recognised by Sommerville et al. [168], who provided 'Operations' that allowed the delegation, decomposition and reallocation of responsibilities. These are similar to the concept of refactoring in software engineering [62], as the intent is to perform a specific change to one part of a model without affecting the model as a whole.

In order to provide a simple and consistent approach to such model changes axiomatic transformations are defined - transformations that do not affect parts of the model other than those being explicitly modified. Using these transformations allows modellers to change specific parts of the model with the confidence that their changes will not unintentionally affect other, unrelated areas of the model.

These axioms define rules for modifying entities that preserve their relations with other entities by providing instructions on how to convert each type of relation involving the entity under the particular transformation. Some transformations are reversible, but others are not; this depends on whether information regarding the responsibility structure can be preserved by the transformation.

We define these transformations for decomposition, aggregation and delegation; the reallocation defined by Sommerville et al. [168] has effectively the same semantics as delegation in formalised responsibility modelling.

Responsibilities can be substituted by decomposing them in full into sub-responsibilities or by aggregating them into higher-level responsibilities; it should be possible to make this transformation on individual responsibilities while leaving the rest of the model unchanged. However, the responsibilities being decomposed or aggregated may be linked to other entities

For responsibilities R, S, R1, R2, Resource X and Actor A; where responsibility R is decomposed into responsibilities R1 & R2 the axiomatic transformation for each type of relation affecting the decomposed responsibility R:

Relation	Declarative Notation	Constraint Notation
Production	R -> X ⇔ R1 -> X, R2 -> X	X:: R ⇔ X:: R1 & R2
Consumption	X -> R ⇔ X -> R1, X -> R2	R:: X ⇔ R1:: X, R2:: X
Reqd. Responsibility	R -> S ⇔ R1 -> S, R2 -> S	S:: R ⇔ S:: R1 & R2
Required Actor	A -> R ⇔ A -> R1, A -> R2	R:: A ⇔ R1:: A, R2:: A
Holds	A => R ⇔ A => R1, A => R2	n/a

Table 4.1: Transformation of relations under decomposition

For responsibilities R, S, T, R1, R2, Resources X, Y and Actors A, B; where the responsibilities R1 & R2 are aggregated into responsibility R the axiomatic transformation for each type of relation affecting the aggregated responsibilities R1 & R2:

Relation	Declarative Notation	Constraint Notation
Production	R1 -> X, R2 -> Y ⇒ R -> X, R -> Y	X:: R1, Y:: R2 ⇒ X:: R, Y:: R
Consumption	X -> R1, Y -> R2 ⇒ X -> R, Y -> R	R1:: X, R2:: Y ⇒ R:: X & Y
Reqd. Responsibility	R1 -> S, R2 -> T ⇒ R -> S, R -> T	S:: R1, T:: R2 ⇒ S:: R & S
Required Actor	A -> R1, B -> R2 ⇒ A -> R, B-> R	R1:: A, R2:: B ⇒ R:: A & B
Holds	A => R1, B => R2 ⇒ A => R, B => R	n/a

Table 4.2: Transformation of relations under aggregation

by relations, and so these relations must be modified to refer to the new responsibilities. Table 4.1 shows the transformation rules for decomposition.

As shown in the table, all relations involving a responsibility that is to be decomposed are transferred jointly and equally to the resulting new responsibilities. Decomposition itself cannot be used to divide up operations such as resource production - a decomposition of responsibility structure does not affect the mechanics of actually producing a resource. Decomposition is fully reversible *as long as the definition is strictly followed*; but is only valid if the decomposed responsibilities share the same exact set of related entities. If they do not (for example, the responsibility R2 has been modified to produce a different resource from responsibility R1) then the transformation is not reversible, as it would be impossible to distinguish the differences in resource production once they were combined.

More generally it is possible to aggregate multiple responsibilities into one larger responsibility. However, this process is not reversible - the original responsibilities can no longer be distinguished once they have been merged, and this makes it impossible to divide them up according to the original structure. Table 4.2 shows the transformation rules for aggregation.

Aggregation behaves similarly to decomposition, with the new aggregate responsibility replacing the original responsibilities in all relations and the aggregate responsibility consisting

of a conjunction of all the original responsibilities.

Decomposition and aggregation have relatively limited effects on the wider model, such as increasing the measured load on an actor when their responsibilities are decomposed into more numerous responsibilities. Important changes can occur when the model is rationalised or adapted after performing decomposition or aggregation - for example, a resource production responsibility that is decomposed may then have different constraints defined for each responsibility, which will affect the likelihood of the resource being successfully produced. However, these changes do not occur directly as a result of the decomposition.

For delegation of responsibilities the 'holds' relation is transferred from the old actor to the new actor, and all other relations between the actor and responsibility remain the same. This leaves no explicit link between the old actor and the new actor; if the old actor is only linked to a responsibility by a holds relation it is possible to make a 'clean break' by delegating it away. This reflects the new structure of relations in formalised responsibility modelling - required Actor relations indicate a functional requirement between the responsibility and the actor and so cannot be changed by delegation; delegation can only change the accountability indicated by the holds relation.

The implications of delegation arise when these existing relations are not compatible with the change of structure, or where this change exposes vulnerabilities. For example, if the delegated responsibility is required for another responsibility still held by the original actor then this has the effect of creating a dependency between the old actor and the new actor, which can lead to a vulnerability. This type of vulnerability can be detected by reliance analysis.

4.11 Applications

Responsibility modelling can be applied at two main levels - requirements analysis and system analysis. When used at the requirements level, responsibility modelling can produce sets of responsibilities, resources and relevant actors based on business requirements, regulation and legal requirements and technical proposals. At the requirements level the model may consist of unrelated fragments that are not directly linked - for example, there may be a legal requirement for a pilot to sign-off on a cargo manifest, and a business requirement for cargo containers to be scanned as they loaded onto the aircraft. These two requirements are completely separate, but are both in the scope of an aircraft cargo management system.

At the requirements stage, models can be used to visualise the problem domain, identify potential flaws in the stated requirements and engage with stakeholders. Incompleteness is natural at this level of modelling, and responsibility modelling can be used to discuss and examine issues of concern. This is similar to the original motivation for responsibility

modelling; the changes made during formalisation do not significantly reduce the ability to use responsibility modelling as technique for informal discussion and analysis. At this stage, automated analysis techniques can be used and can deliver some useful results (such as highlighting areas of the requirements that are too vague), but their results require additional interpretation to detect false warnings that will be resolved during later development.

At the system analysis level the responsibility modeller should aim for completeness. By this stage of development the system scope and model structure should be decided, and hence any ambiguity or omissions in the responsibility model represent ambiguities and omissions in the system structure itself. Separated or unattached sections indicate a lack of integration within the system, or that certain elements are not related to the core duties of the system.

Many process and structure models (in all types of domains) contain unformalised behaviour that is nonetheless implied or referred to by the formal element of the model. This may be an explicit decision to keep parts of the system out of scope or may reflect uncertainty over the use of that part in practice. In some models these unformalised elements are clearly signposted; however, many models do not distinguish them clearly. This opens up the possibility of inconsistent application of the model and important elements left undone due to their non-formalised nature. Responsibility modelling can clearly identify non-formalised or ambiguous formalised elements, allowing clear discussion of potential issues. Equally, where these variations are not pertinent to a discussion responsibility modelling provides for convenient abstraction.

Responsibility modelling is especially appealing when comparing deployments or implementations against the theoretical standard. The complexity and variation of social-technical systems means that many real systems vary significantly from their conceptual model, but these differences may not be modelled. As a result, analysis is often performed on abstract models of the system that do not represent real-world usage, leading to analysis that does not capture actual behaviour and provides a false sense of security.

Responsibility modelling can potentially alleviate this by allowing both high-level and implementation-level models of the system to be produced using the same notation, which will allow easier comparison of the two views. Ideally, these multiple views of the system could be mapped together, allowing direct checking of the implementation against requirements. However, there are ambiguities in the potential semantics of this relationship, which are discussed in Section 9.5.

4.12 Modelling Examples

This section illustrates the use of responsibility models to construct representations of the structure of socio-technical systems. First a basic model of the meeting scheduler system

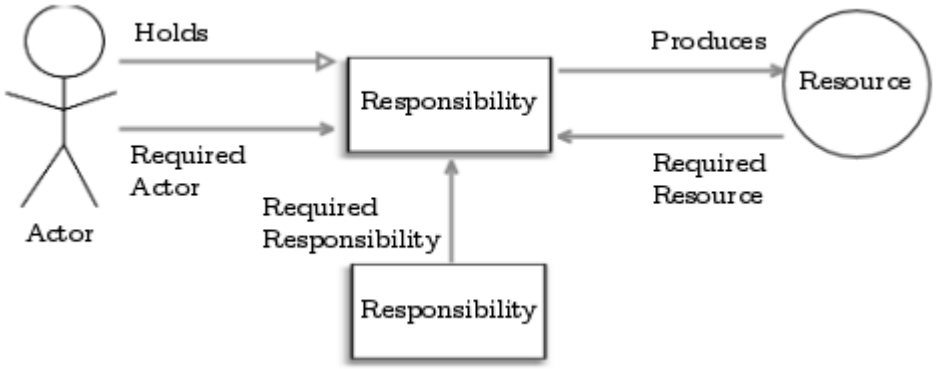


Figure 4.7: Labelled key to formalised responsibility modelling notation

(as defined by van Lamsweerde et al. [189]) is used to introduce examples of the notation, before considering a more complex model of the Scrum agile development team structure described by Schwaber and Beedle [155]. These examples are used to demonstrate the modelling notation and elements of the constraint language and show how they can be used to model systems of moderate complexity. Some informal evaluation is performed, but formal analysis is not demonstrated here and is the subject of Chapter 5. A recap of the formalised responsibility modelling notation is shown as a labelled diagram in Figure 4.7.

4.12.1 Meeting Scheduler

The meeting scheduler as defined by van Lamsweerde et al. [188] is a common socio-technical case study, and has been modelled using KAOS [189] and i* [202]. Figure 4.8 shows a responsibility model of the scheduler using graphical responsibility modelling notation.

The Meeting Scheduler System is intended to support the organisation of meetings. Meetings are arranged in the following way. A meeting initiator asks potential attendees to list dates within some range that they would prefer to attend a meeting (the preference set) and dates where they cannot attend (the exclusion set). Additional preferences may be expressed by attendees, such as requirements for special equipment or choices about the meeting location. A proposed meeting date should belong to as many preference sets as possible, and not be a member of the exclusion sets; if this cannot be achieved, a date conflict has occurred.

The model proposes a system structure containing four distinct roles. The meeting initiator is responsible for defining the list of participants and initially setting up the meeting; the meeting scheduler system then performs the practicalities, such as determining the time and location. Meeting participants provide their availability times, expressed as the preference set and exclusion set, and can also specify equipment requirements. Additionally, important participants can also specify their preferences for the location.

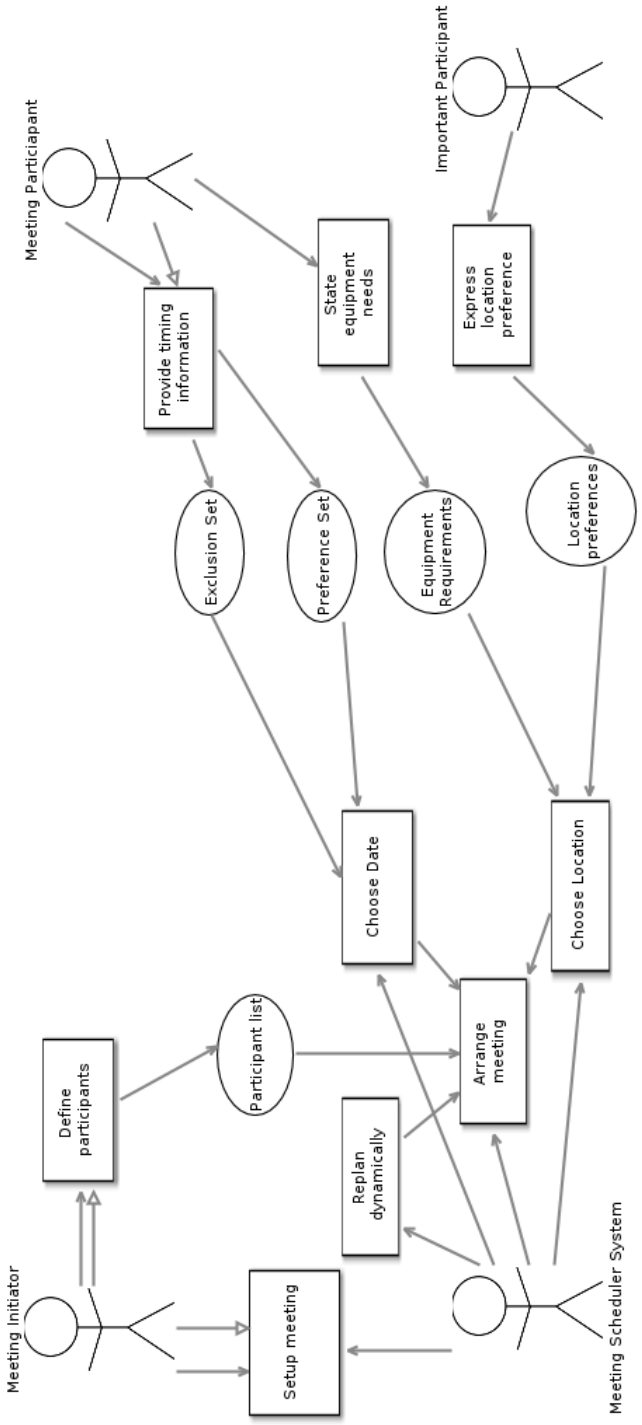


Figure 4.8: Responsibility model of the meeting scheduler described by van Lamsweerde et al. [188]

The particular version of the system described by this model contains some important subtleties. Of the three types of information that participants (either normal or important) can provide only one of them is held by the corresponding actor. This implies that there is no obligation that participants provide information other than timing information; this is consistent with the definition that states that providing these types of information is optional. However, responsibilities of the meeting scheduler (specifically, Choose Location) depend on the availability of expressed location preferences and equipment needs, and so would not be discharged if no preferences are provided. This problem is not addressed in the original definition - it is unclear how the location is determined, and if factors other than preferences are taken into account.

The Meeting Scheduler System also holds no responsibilities itself, and is merely the required actor for the responsibilities that are held by others to be discharged. This expresses a particular view of system deployment - the scheduler is a piece of software, and therefore cannot be held to the same standards of accountability as a human or an organisation. However, this means that a responsibility failure due to the scheduler cannot be traced or attributed at all, which may be unsatisfactory. Alternatively, the scheduler could be provided with 'holds' relations, which would allow failures to be directly attributed (although this may not be meaningful, for the reasons mentioned earlier). Another option would be to introduce an additional actor, the system developer or maintainer, which could hold a responsibility for ensuring the scheduler operates correctly.

This example also demonstrates some of the difficulties in effectively dividing up responsibilities in manageable units. 'Choose Date' and 'Choose Location' are both sub-responsibilities of 'Arrange meeting', which initially appears to be a sensible division of two important parts of arranging a meeting, given that they both require different sources of information. However, both responsibilities are really dependent on the same resource and are irreversibly interlinked - the availability of certain locations is presumably time dependent, so separately selecting time and location (as this model describes) could lead to arranging meetings where the location is unavailable. At this level of abstraction the use of two separate responsibilities does not indicate that the selection of date and location must be completely independent (the actual logic for selection is not defined), but does also not specify that they must be connected (which is necessary to avoid potential conflicts).

The role-based nature of actors is also clearly demonstrated in this model. The separate responsibilities for Meeting Participant and Important Participant do not imply that individuals who are important participants cannot provide timing information and state equipment needs; such a real-world individual fulfils two responsibility modelling roles.

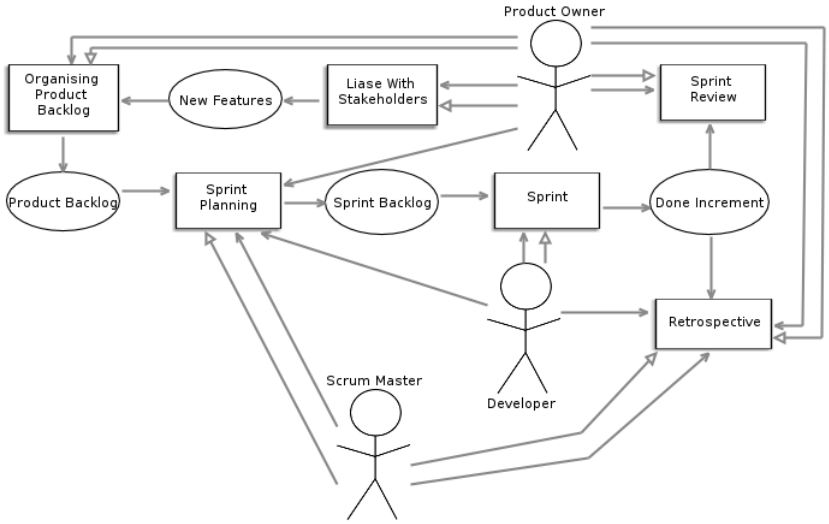


Figure 4.9: Responsibility Model of Scrum

4.12.2 Scrum

The Scrum process is an agile development framework for software development [155]. Scrum is an iterative process, where the product is developed in a series of sprints, and requirements are expressed by a product owner. Additionally, the Scrum team lacks a hierarchy and does not contain managers or assigned roles and groupings. Instead all team members take part in all Scrum activities, which are overseen by a Scrum Master who acts as a facilitator. Our model of the responsibilities in the Scrum process are illustrated in Figure 4.9.

The figure shows that work is organised around responsibilities for planning, undertaking and reviewing a sprint. The iterative structure of the responsibilities (and thus the overall process) is evident in the model, in contrast to more linear models of development.

Additionally, the model shows that Scrum is a collaborative process where all roles collaborate in many different responsibilities in different ways. For example, Developers operate in collaboration with the Scrum Master and Product Owner for Sprint Planning, in which the set of objectives for the next Sprint is decided. Similarly, all actors collaborate in the conduct of the Retrospective and Sprint Review, although they each perform different elements of the process.

Note that this example demonstrates that actors are roles, not individuals; in Scrum, it is common for the Scrum Master to also be a Developer. As a result, extra care (ideally aided by tool support) is necessary when considering the risk of overload, as one actual individual or organisation may be acting in a number of different roles.

The model also shows the difference between being required for a responsibility and being responsible for it. In most cases, actors are both required by and responsible for (holding) certain responsibilities. For example, the Product Owner is required to complete the Sprint

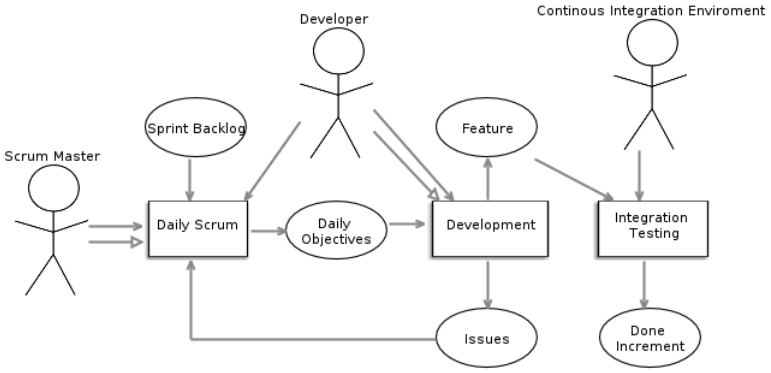


Figure 4.10: Refinement of 'Sprint' Responsibility.

Review, and is also accountable for its success or failure. However, some other responsibilities are only held by one actor. For example, the Scrum Master is the sole actor to hold Sprint Planning and Retrospective; this indicates they they alone hold ultimate responsibility for ensuring its completion. These responsibilities still require other actors to be successfully completed, and so the Scrum Master must rely on the Product Owner and Developer in order to fulfil their duties.

A responsibility can be decomposed in order to understand how agents collaborate to discharge the overall responsibility. Figure 4.10 illustrates how the Sprint responsibility is decomposed to show the assignment of sub responsibilities. The Developers retain responsibility for undertaking development of new features or remedy of features, as prioritised during the Daily Scrum. However, the Scrum Master takes responsibility for coordinating the Daily Scrum. The exact division of responsibilities within the Daily Scrum may be identified by further refinement if desired.

The figure also illustrates the modelling of heterogeneous human, organisational and technical components consistently as agents with responsibilities. The development team is an organisational agent consisting of several (unidentified) developers. Responsibility for integration testing is delegated to a Continuous Integration Environment, a software application that is configured to monitor for changes to the target system's code base.

The decomposed model is consistent with the overall model presented in Figure 4.9. Resources that are inputs and outputs to the Sprint responsibility (Sprint Backlog and Done Increment respectively) are similarly represented as boundary elements for the decomposed diagram. Similarly, the agents that hold the overall responsibility for the Sprint (Developer and Scrum Master) are present in the decomposed diagram. The Continuous Integration environment does not hold responsibility for conducting the overall Sprint, so only appears in the sub-diagram in association with its specific responsibilities. Depending on the modeller's preference and the target audience, these refinements can either be presented separately, or used to expand the core responsibility model.

This initial model of Scrum can also be expanded to describe more complex behaviour using the new additions to the responsibility modelling notation. For example, Development is currently discharged and features and issues produced as long as the developer is active and daily objectives are available. This is an idealised representation of software development, as numerous reasons can lead to seemingly well-planned development failing to complete. We can explicitly model this by introducing uncertainty into the definition of Development, with the following satisfaction criteria:

Development:: Developer Active & Daily Objectives Exists & ?

In this version of the system the activity of the developer and availability of the objectives are still necessary to successfully discharge development, but they are not necessarily sufficient - this definition of Development can still fail, even when both requirements are correct. This then weakens the strength of claims that can be made about the system as a whole, as it is no longer possible to guarantee complete and successful discharge of all responsibilities.

Additionally, responsibilities can also be augmented with constraint logic to indicate more complex relations. Currently, the Daily Scrum requires two resources - Sprint Backlog (representing work remaining) and Issues (representing new problems that have been detected), which are then used to plan the objectives for the next phase. However, there are situations where only one of these inputs may be required - for example, when no issues are raised during a particular day's development. As currently expressed in the model this could lead to inappropriate failures - the scrum is dependent on Development (which produces the Issues), but it might in practice be possible to perform a scrum without associated development (such as if dividing up a backlog). Instead, the Daily Scrum responsibility could require one of either Sprint Backlog or Daily Objectives, and this can be expressed with the following constraint logic:

Daily Scrum:: Developer Active & (Sprint Backlog Exists || Issues Exists)

This version more accurately describes certain edge cases by still discharging successfully when one resource is unavailable. However, it now means that the Scrum always succeeds as long as the developer is active and one resource is available, while there are some scenarios where this might be inappropriate (such as when there are live issues discovered by developers, but the Scrum Master forgets to discuss them). In effect, removing 'false failures' may now introduce 'false successes'. This highlights the difficulty of producing models that are both suitably abstract and suitably accurate - specifying the organisational processes in exact detail is complex and potentially contradictory, but more abstract models may provide system definitions that deviate from the intended reality.

This section focused on the initial modelling of scenarios, and simple analysis by inspection of the models' structure. A key contribution of formalised responsibility modelling is that it enables more rigorous and complex analysis to be performed, based on an understanding of

the formal semantics. This type of analysis is discussed further in the following chapter.

4.13 Conclusion

This chapter has presented a consistent definition of responsibility modelling by formally defining the semantics of entities and relations. Entities retain the same meanings as in previous responsibility modelling systems, but with slight changes to the notation. Relations are restructured to provide clearer modelling - sub-responsibilities are generalised into required resource relationships, and actor assignment is separated into two distinct relations - the required actor relation (which models the direct need for a particular actor) and the holds relation (which models the responsibility / accountability relationship). These changes lead to a reworking of traditional responsibility modelling concepts such as delegation and dependency.

The inability of responsibility modelling to handle complex relations between entities is addressed by the introduction of satisfaction criteria - a simple logic for expressing additional details in relations where appropriate. This allows explicit modelling of common relations such as alternative implementation options and fallback systems.

This logic is then used to introduce an explicit method for capturing uncertainty in models. Socio-technical system models often make the assumption that they completely represent the underlying problem domain, even if requirements capture has left some parts of the model partly or completely unspecified. By tagging any entity with an uncertainty marker these gaps can be identified and modelled, distinguishing between intentional and unintentional uncertainty. In addition, this explicit uncertainty redefines analysis by identifying areas where modelling cannot accurately analyse the system due to a lack of certainty.

The temporal nature of responsibility modelling is also more carefully examined. Models have always featured implicit temporal elements (a resource must be produced before it can be consumed), but the detailed meaning was left undefined. We therefore define the principle that responsibilities are ‘eventually’ discharged - that responsibilities must be completed within an arbitrary time period appropriate to the problem scenario. This eliminates the need for careful evaluation of discharge times and removes the potential for race conditions. Transformations are also defined for common modelling changes to eliminate unintentional effects on other parts of the model.

Short analyses of methodological issues such as model scope and abstraction are given, before this chapter concludes by demonstrating the full range of the revised notation in small-scale worked examples of a meeting scheduling system and the Scrum agile development process.

Chapter 5

Analysis

5.1 Introduction

In the previous chapter we presented a consistent, formalised notation for responsibility modelling. However, modelling socio-technical systems is a means to an end, rather than a complete task in itself. Socio-technical models are used to analyse systems on criteria including safety, security, performance and usability [13, 109, 112, 181]. In previous responsibility modelling studies, analysis has consisted of informal or semi-formal inspections of the model and semi-formal analysis using guidewords. The key advantage of formalising responsibility modelling is that analysis can be performed on a much more structured basis, and that different forms of analysis can be automated according to set rules.

This chapter describes both formal and informal analysis techniques developed during this research. Firstly, the RESME toolkit is introduced and described. This Eclipse-based toolkit provides a graphical interface for the creation, editing and analysis of responsibility models; in particular, it provides full support for automatic analysis techniques. Formal analysis consists of five different automated techniques: overload detection to identify overstressed actors; assignment checks to locate unheld responsibilities and unproduced resources; reliance analysis to identify inter-dependencies between different actors holding related responsibilities; criticality analysis to identify the most vulnerable single points of failure and full discharge analysis, which evaluates the discharge and production of entities according to specified satisfaction criteria.

The informal approaches are the use of patterns and anti-patterns to recognise responsibility structures and potential vulnerabilities and the identification of common modelling problems. Additional semi-formal approaches include role analysis to identify potential overloads and conflicts of interest caused by one individual holding multiple actor roles and the use of what-if analysis to selectively disable certain model entities and examine the knock-effects on

other parts of the model.

5.2 RESME Toolkit

Tool support is desirable for modelling of non-trivial systems, allowing for the automatic or semi-automatic application of analysis and proof techniques that would be time-consuming and impractical to perform by hand [203]. Previously, responsibility models were constructed with general purpose imaging programs (e.g. [12, 169]) or with the aid of plugins to general purpose modelling tools such as Microsoft Visio (e.g. [174]). These plugins included a definition of the graphical notation, providing consistency, but did not enable any form of automated analysis.

To address this we have developed the RESME toolkit [157] to support the construction and analysis of formalised responsibility models. This toolkit provides a user interface for graphically constructing new responsibility models and then applying a range of automated analysis techniques. The user interface of the toolkit is shown in Figure 5.1.

The RESME toolkit is used as the primary mechanism for implementing automated analysis, and is used to both construct and present responsibility models for the case studies in the following chapters. Given the importance of the tool in these case studies, this section provides an overview of the key features, interfaces and implementation details of the RESME toolkit.

The RESME toolkit is implemented using the Eclipse Sirius framework [49] and the Eclipse Modelling Framework (EMF) [50] . The EMF is used to model the underlying data structures (entities and relations) and generate corresponding Java code from this model; the generated classes from this process are then extended with hand written functions that perform model-wide analysis. It also provides a cross-platform XML representation of user-generated responsibility models. Sirius provides a standard graphical editor for EMF models which enables the visual representation of models and simple drag-and-drop editing; complex editing details and model validation is performed by customised rules defined in the Acceleo [51] model transformation language. Finally, the ANTLR [139] parser system is used to define and parse the satisfaction criteria logic, with evaluation performed by native Java code.

This toolkit is implemented as Eclipse plugins grouped into an Eclipse feature, which allows easy deployment using the Eclipse update-site mechanism. This allows the toolkit to be installed into any existing Eclipse installation without the need to manually configure dependencies or settings. A ready-to-deploy package of the toolkit is hosted as an Eclipse update site [163] and the full source code and version history is available as a GitHub project [157].

Models can be constructed and edited graphically, with all standard entity and relation types supported. In addition, several special purpose entities and relations have been added to

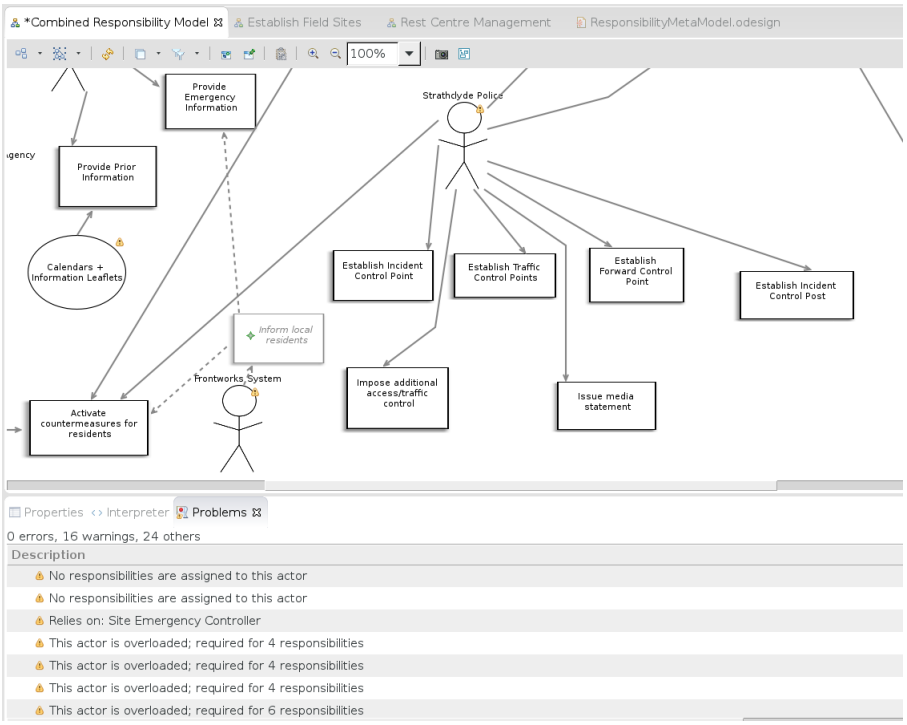


Figure 5.1: Analysis in the RESME toolkit, showing reliance, overload and selectively disabled responsibilities

support modelling extensions developed for specific case studies. Models are constructed by first dragging entities from the toolbar and placing them in the main modelling screen; relations can then be added by selecting the appropriate relation type from the toolbar and then selecting the two entities to be linked. Details for each entity can be viewed and edited using a tabbed detail browser. The visual layout of the model is purely for aesthetic purposes. The automated layout system is usually capable of producing an uncluttered diagram, or a user may apply their own manually. Alternative tree and table representations are also available, which provide a clear way of viewing entities but obscure the source and destination of relations.

Large responsibility models can be complex and confusing, and so the toolkit offers several tools to make them easier to understand. Extensions to the basic set of entities and relations can be toggled on and off using layers; when a layer is disabled any entities or relations of that type are removed from the diagram and from the toolbar, but remain unchanged in the underlying model.

Models can also be structured using sections, which are used to display and edit specific parts of the system. Sections can contain any number of entities and relations, but due to tooling restrictions, each entity can only be within one section. These sections can be displayed separately using Section diagrams and changes made in section diagrams are automatically transferred to the underlying model and the main responsibility diagram. This enables visualisation of model sub-areas, and can also be used to construct a model incrementally out

of sections. The use of sections is purely to divide up visual representations of the model, and does not divide the model itself - for this entirely separate model files should be used.

RESME also provides partial support for realisation relationships and multi-level modelling via the instance layer. This enables the addition of instance actors to the model, which can be linked to normal actors using realisation relations. This relation indicates that the instance actor is a implementation of or special case of the original actor. The more general concept of instantiation across all entity types is not supported in the current release.

Satisfaction criteria can be specified for any entity using the properties view. Satisfaction criteria are not visually represented - any existing relations remain displayed, and new links introduced only in satisfaction criteria are not displayed. Likewise, local forms of analysis are not affected by specifying satisfaction criteria, and still operate based on the standard definitions of relations. Satisfaction criteria are used to modify the results of satisfaction analysis, which uses a naive evaluator by default (all required entities must be available, and there are no additional requirements); each entity that has custom satisfaction criteria defined uses these in place of the default.

Explicit uncertainty in satisfaction criteria is handled by the generation of warnings on each entity containing uncertainty. The effects of this uncertainty are not automatically analysed, but instead draw the attention of the modeller to the entities they need to examine. This provides a quick overview of potential sources of wider system uncertainty, but requires the modeller to manually consider the effects of uncertainty propagating throughout the model, which may induce additional vulnerabilities.

5.3 Automatic Analysis Methods

The existence of a formal structure for responsibility modelling enables wide ranging analysis to be performed automatically and semi-automatically on models that adhere to the formal structure. This section presents five distinct analysis techniques that benefit from this formalisation or that would not be feasible without the use of computerised tool support. These techniques include both methods that rely on local analysis (such as analysing the direct relations of an entity) and methods that analyse the model (such as the identification of critical points of failure). These techniques are static analysis methods that identify potential vulnerabilities by analysing the structure of a responsibility model.

For each technique the motivation and basic principles are introduced, the main applications described, common results discussed and results that may require special interpretation considered. The identification of potential vulnerabilities by automatic analysis may indicate either a flaw in the actual domain or a flaw in the model, so care must be taken to fully

understand the implications of any result. This section defines and introduces these methods; we leave validation to case studies in later chapters.

Sommerville [166] discussed eight different forms of responsibility vulnerability in an informal manner, six of which have corresponding automatic analysis techniques in formalised responsibility modelling. Unassignment of actors and lack of resources are directly implemented, as is overload. Sommerville's 'fragility' is essentially identical to our concept of criticality - identifying the most vulnerable parts of a system. Sommerville's 'lack of authority' can be analysed using formalised responsibility modelling's reliance analysis, as can their concern over conflicting authorities for actors. Duplication of duties and uncommunicated assignment are the only analysis forms that are not automated; these are difficult to identify automatically because they are not directly represented in the responsibility structure.

All these forms of analysis are supported by the RESME toolkit. Local (overload, assignment, reliance) checks are performed by validating the model file (invoking validation applies all local checks to all entities simultaneously) while global (criticality, discharge) checks are triggered by selecting the appropriate analysis method from the toolbar and applying it to the model. In both cases the results of the analysis are presented in the standard Eclipse format as warnings or information points; these are shown as icons with tooltips on the graphical representation of each entity, and are listed in full in the Problems view.

5.3.1 Overload

Failures in socio-technical systems can occur when parts of the system are stressed or overloaded - in particular, individuals and organisations may struggle when attempting to deal with unexpectedly large or complex tasks, or simply because they have to handle more duties than they can give full attention to. For example, Adams et al. [2] describe how pilots lost situational awareness when attempting to process multiple tasks simultaneously. Adler and Benbunan-Fich [3] show that increased multitasking leads to a significant loss in task accuracy. These overloaded actors could be relieved by assigning some of their responsibilities elsewhere, or by providing them with additional resources, but only if they are correctly identified.

Thresholds can be set to determine potentially overloaded actors that are required for an excessive amount of responsibilities, potentially leading to degraded operation. Overload on actors indicates either a risk due to multiple significant responsibilities potentially requiring to be discharged simultaneously or due to the actor being required to oversee or supervise a large number of responsibilities. The thresholds for these types of risk are generally quite different - discharging more than 2 or 3 responsibilities simultaneously is likely to lead to problems as discharging tasks simultaneously requires more effort than the total of discharging them

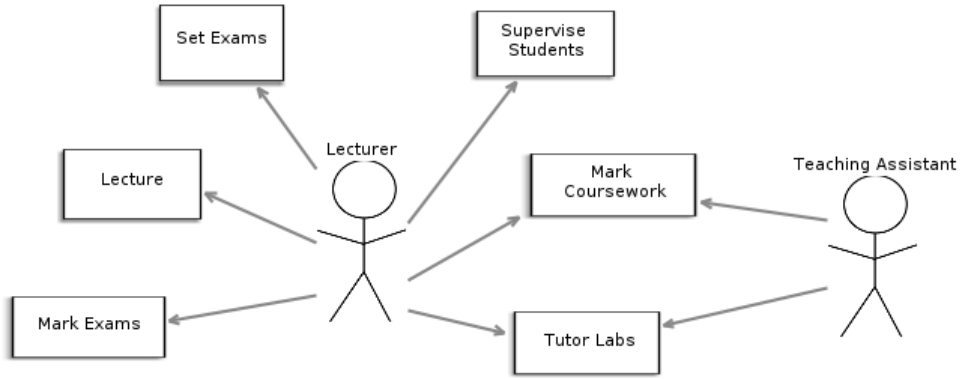


Figure 5.2: Example responsibility model demonstrating overload

separately [97], while an actor may be able to manage substantially more responsibilities if they are less complex or do not require direct action.

This makes the concept of overload in responsibility modelling more complex than in a task- or process-based modelling system. Actors may be overloaded if they are required to discharge multiple responsibilities concurrently, as in task-based modelling. However, actors may also be overloaded if they are required for a large number of distinct responsibilities, even if they play only a minor role in the discharge of these responsibilities. This 'cognitive' overload occurs as actors are required to continuously track and evaluate the responsibilities for which they are required. The abstraction inherent to the concept of responsibilities means that this distinction cannot be formally analysed, and so these two distinct forms of overload are represented as one metric.

Formally, an actor is overloaded if they are required for a number of responsibilities above a set threshold, which was initially set to four. This value was selected to exclude actors holding several responsibilities from being declared overloaded in order to avoid excessive false positives. The reliability of this metric is improved by modelling responsibilities at consistent levels of abstraction to ensure that the effective loads are comparable.

Figure 5.2 shows a simple responsibility model for teaching responsibilities within a university. The Lecturer is required for six different responsibilities, while the Teaching Assistant is only required for two. As a result, the Lecturer is an overloaded actor, as the number of responsibilities they are required for is above the threshold. The fact that two of the responsibilities are shared (they require both the Lecturer and the Teaching Assistant) does not affect the results of overload analysis, as is not possible at the responsibility level to determine how the effort of these responsibilities is divided.

The generation of an overload warning is not definite evidence of a vulnerability, but a warning sign that requires further investigation. A model is likely to feature responsibilities across different scales and different levels of importance, and holding a large number of 'small scale'

responsibilities need not be more demanding than holding fewer, larger responsibilities.

This can be addressed by specifying the load value of responsibilities - representing the amount of attention and effort required to discharge them. By default, all responsibilities have a standard load value of 1; load values are relative to other responsibilities, so increasing the load to 3 indicates a responsibility that places three times the standard load on its respective actors. This has a corresponding effect on overload warnings; it is easily possible for an actor holding one responsibility to be overloaded if that responsibility has a particularly high load value.

However, care should be taken not to over-specify responsibility loads. The load value is a broad representation of the complexity of the responsibility, not an exact specification of the time taken or resources required. Overload detection is an advisory warning, and so minute differences in load make little difference; differences in load values are best used for responsibilities that have appreciable differences in complexity within the context of the model.

Overload detection is most effective when a model of a problem domain represents responsibilities and actors at the same level of abstraction. For example, a police force is responsible for maintaining law and order, whereas a traffic officer who is a member of that police force is responsible for enforcing traffic laws. Misleading results can occur if these two levels of abstraction are mixed. For example, a police service could be modelled as being responsible for the traffic laws, investigating cyber crime, responding to emergency calls, etc. Overload analysis would consider the police service overloaded, whereas in practice this is an effect of inappropriate mixing of abstractions. This is best addressed by aiming to maintain the same level of abstraction across the entire model, but like all forms of automated analysis a certain level of human judgement is required to interpret the results.

5.3.2 Unassignment

The correct operation of a system depends on the availability of resources and actors - the failure to produce resources via the discharge of responsibilities is a vulnerability within the model of the system. A full analysis of resource production or responsibility discharge can be complex, especially when complex satisfaction constraints are defined. However, a subset of potential vulnerabilities can be detected using unassignment checks.

Unassignment warnings are generated for each resource that does not have an associated Production relation, and for each responsibility that does not have a Required Actor relation. Relations to disabled entities are excluded, but no other checks are made as to the viability of the related entities.

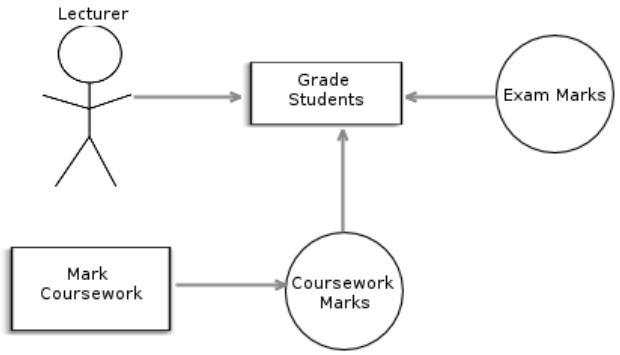


Figure 5.3: Example responsibility model showing unassigned entities

Figure 5.3 shows a partial responsibility model for assessing university students. Grading students requires a Lecturer, Exam Marks and Coursework Marks. Unassignment warnings are generated for Mark Coursework and Exam Marks; Mark Coursework has no Required Actors, while Exam Marks is not produced by any responsibility. However, no warning is generated for Coursework Marks, as these are produced by Mark Coursework, even though Mark Coursework itself is subject to a warning. Unassignment checks are local to each entity, and do not check further than their direct relations. Instead, a full-system wide analysis can be performed using discharge detection.

In most cases, unassignment indicates that the resource cannot be produced or the responsibility cannot be discharged; this can reflect an issue in the modelling process (an attempt to limit the model scope, or an entity missed out) or an oversight in the actual system, such as an undocumented process or an assumption of constant availability.

5.3.3 Reliance

Interactions and interdependency occur routinely between actors in socio-technical systems. Actors can share certain responsibilities, where multiple actors must collaborate in order to achieve a common goal. These types of relationship are observed in a responsibility model by looking at the requirements for a particular responsibility.

However, a more subtle form of interdependency occurs when an actor indirectly relies upon another in order to be able to discharge their own responsibilities. For example, an actor may hold a responsibility that requires a resource in order to be discharged, and so the actor's responsibility can only be discharged if the resource is successfully produced. If the resource is produced through the discharge of a responsibility held by a different actor, the original actor relies on the second actor. Without them the original actor cannot discharge its own responsibilities. This places actors in a difficult position - their own success in discharging responsibilities can be reliant on those they have no direct control over.

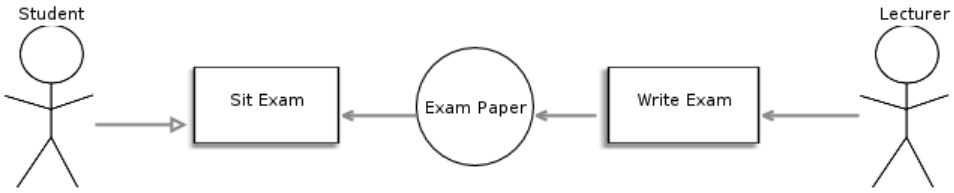


Figure 5.4: Reliance example - the Student relies on the Lecturer

In particular, this dependency produces problems when entities hold responsibilities where the activity of the work is performed completely by another agent. This can easily occur when there is uncertainty about the responsibilities of particular agents, or where work is delegated away. For example, drivers may blame the local authorities for the state of the roads, even if these are trunk roads managed by central government. In such a case it can be said that the local authority holds a responsibility to manage the roads, which itself can be refined into local roads (requiring the local authority) and major roads (requiring central government). As a result, the local authority is dependent on central government to discharge its responsibilities, but has no ability to influence them.

Despite this, dependency is a core part of most large systems. Where dependency relationships match organisational hierarchies or contractual obligations the system has the required control mechanisms to successfully discharge its responsibilities. Dependency only leads to vulnerabilities where there is a mismatch between the relationships implied by the responsibility structure and the actual relationships within the system.

Reliance analysis generates a corresponding set of actors for each actor within a responsibility model. This set contains all other actors that are directly or indirectly required by the responsibilities held by the original actor; this includes both direct relations (i.e. a required Actor relation) and indirect relations (where a responsibility requires another entity, which directly or indirectly relies on another actor).

Figure 5.4 shows an example model displaying the use of reliance analysis. A Student holds the responsibility to Sit an Exam, but to discharge this they need the Exam Paper resource. This resource is produced by the responsibility Write Exam, which is held by a Lecturer. The Lecturer is required by a responsibility that is indirectly required by a responsibility held by the Student, so the Student therefore relies on the Lecturer. Without the Lecturer there would be no Exam Paper to be sat, and so the Student would be unable to discharge their responsibility.

This form of reliance can be a source of potential vulnerabilities. Actors can be placed in a situation where they are held accountable for responsibilities that they cannot discharge by themselves; they are expected to discharge their own responsibilities regardless of the necessity of external support. This is particularly problematic where reliance crosses organisational

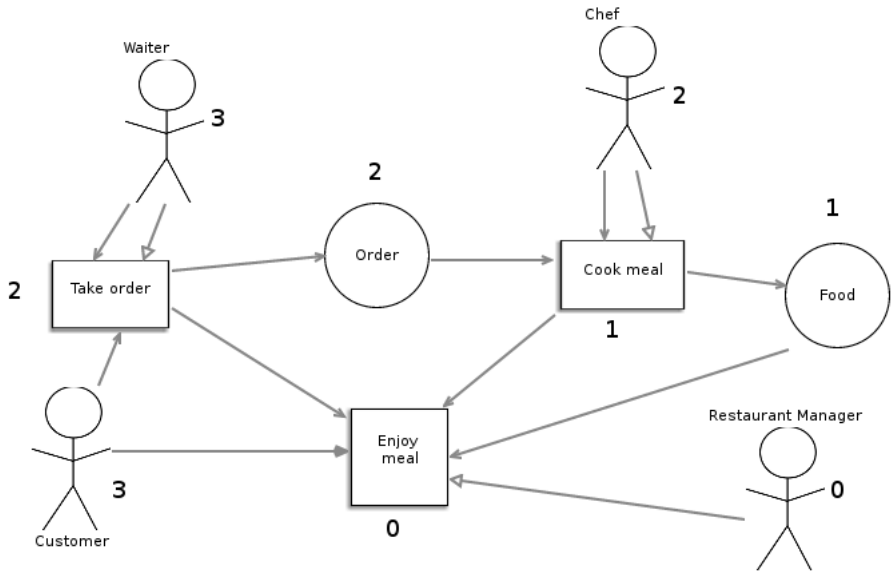


Figure 5.5: Example responsibility model annotated with criticality scores

boundaries, as this leaves actors without the 'soft influence' on others that they have by virtue of being within the same organisation [166].

The set of actors relied upon can be compared to management structures. If an actor relies on another actor that they manage or supervise it is more reasonable to blame them for failings than if they have no control over the second actor. For example, the manager of a team is likely to rely on the members of their team in order to discharge all their responsibilities, but they have the authority to ensure that the team members act as required. If an actor relies on others over whom they have no authority there is little the actor can do to ensure that what they require is done, and the appearance of such relationships is a sign that the model lacks an appropriate authority structure.

5.3.4 Criticality

Two forms of analysis are also possible which operate across the entire model. Criticality analysis detects the most critical entities - those that contribute the most to the system and would cause the highest number of responsibility failures if they were non-functional.

Criticality is calculated by identifying the number of responsibilities that depend on an entity in order to be discharged, such as needing a resource or requiring other responsibilities. This is calculated by traversing the graph of all entities in the model. The criticality score of a responsibility is the sum of the criticality score of all entities that depend on it. The criticality score indicates the number of other responsibilities that would not be discharged if the critical entity was unavailable.

Figure 5.5 shows a responsibility model of serving food in a restaurant, annotated with the criticality score of each entity. The most critical actors are the Waiter and Customer (tied with a score of 3 each), the most critical actor resource is the Order and the the most critical responsibility is Take Order. Criticality scores are calculated by evaluating the dependencies of each responsibility; for example, Cook Meal is required for Enjoy Meal (which gives Cook Meal a criticality score of 1), Order is required for Cook Meal which is itself required for Enjoy Meal (giving Order a criticality score of 2) and so on. Criticality considers only required entities and not Holds relations, so the Restaurant Manager has a score of 0, as they hold a responsibility but are not required by it.

In practice, there are two main cases of criticality. Firstly, an entity may receive a high criticality score because it is at the start of a chain - for example, the first resource in a complex production/consumption process, or the root responsibility at a chain of dependencies. Alternatively, a high criticality score may arise simply when an entity is directly involved in a large number of relations.

In either case, the critical entity warning can be used to identify the need for fault-tolerance and redundancy in the system. In a long chain of entities, this redundancy can be provided by allowing for multiple ways to initiate the process (eliminating a single point of failure at the start), and by multiple routes throughout the chain (preventing the chain from failing part way through). If a single entity is involved with a large number of relations this can be addressed by transferring some of their relations to other entities where possible. Alternatively, the entity itself can be made more redundant - for example, responsibilities can be replaced with primary and alternate modes, and actors can be augmented with automated fallback systems.

Criticality analysis relies on a consistent level of detail and abstraction in a model. Each responsibility is considered to have the same level of importance, so a cluster of specific responsibilities is considered more critical than a single overarching responsibility. If one part of the model is specified in more low-level detail than another then the low-level part of the model will contain a higher density of entities, and hence receive a higher criticality score. This can be partially addressed by breaking the model down into sub-models and analysing each one separately; this requires very careful construction in order to avoid ignoring interactions between the different sub-models.

5.3.5 Discharge

The second form of model-wide analysis is responsibility discharge detection, which is augmented with the constraint language described in Section 4.6. By default, responsibilities are considered to be successfully discharged if all required elements (the entities to which they are linked by relations) are active. However, sometimes the discharge criteria for a

responsibility are more complicated. A responsibility may for example be dischargable by either one of two separate actors, or may rely on the availability of a subset of different resources. In these cases, the more complex behaviour can be expressed using the constraint language.

When discharge analysis is performed, initial checks are made on entities without complex constraints to determine if they can be discharged. After this, the constraints defined on complex entities are evaluated by a constraint parser, and responsibilities that fail to discharge are indicated. If combined with the selective disabling of model elements this technique allows for an effective analysis of system failure modes. This allows for the examination of fault tolerance and redundancy within the system, and further automation allows the most serious points of failure to be determined by checking the number of undischarged responsibilities caused when each object in the system is disabled.

5.4 Modelling Problems

Responsibility models may exhibit certain characteristics that are potentially indicative of mistakes in the modelling process, such as missing relations or complex responsibility structures. Understanding these common problems allows the identification and correction of such mistakes easily and in a consistent manner. These problems are akin to the software engineering concept of 'bad smells' - indicators of poor software design practice that may cause difficulties in the future [14]. In systems modelling, bad smells in a model may represent a mistake or a non-optimal modelling choice, but they may also represent a genuine problem or uncertainty in the problem domain. It is therefore important to fully understand any apparent problem before addressing it, as changing a model so that is 'correct' may actually make it less representative of the system it is attempting to describe. In this section, several common modelling problems that may be indicative of either poor modelling or issues in the actual system are described and evaluated. These problems have been observed in the construction of responsibility model for this thesis, as well as in previous responsibility modelling studies.

Ideally, all responsibilities in a model should be at a broadly similar level of abstraction. This helps to make the scope of the model clear, and makes analysis more meaningful, as responsibilities are more likely to be of similar effort and importance. However, it is common for responsibility models to have varying levels of abstraction. This is not inherently a problem, but can indicate several potential causes that require investigation. Firstly, inconsistent abstraction levels may simply represent an example of confused modelling, where the modeller has inadvertently mixed levels of detail without due cause. In this case, a restructuring of the model will provide greater consistency and improve the reliability of analysis results.

Alternatively, confusion around levels of abstraction may represent actual inconsistencies

or wide variance in the domain itself, or at least in the sources of information used to build the model. This may arise from combining sources that take different views of the system (such as operational records vs. regulatory requirements) or because individual sources cover wide-ranging areas and focus in more detail on specific topics. In this case, the inconsistency in the model is inherent to the representation of the system and cannot easily be eliminated; instead, the multi-level structure of the model should be taken into account when performing analysis.

Issues around the abstraction of responsibilities can also have implications for actors. Actors that appear to hold large numbers of responsibilities may only do so because the responsibilities they hold have been decomposed to a high level of detail compared to other responsibilities in the same model. This can potentially be addressed by restructuring the responsibilities as mentioned above; if this structure arises because of complexities in the domain the only option is to carefully consider the results of actor analysis, especially when drawing comparisons between actors that may have responsibilities modelled at different levels of abstraction.

The distinction between the 'holds' relation and the 'required actor' relation can potentially lead to confusion; in particular, actor-responsibility relationships that use one but not the other. Each possible combination represents a different type of interaction; 'holds' indicates accountability and duty; 'required actor' indicates practical need and both 'holds' and 'required actor' indicates both practical participation and accountability. Each option has its uses, and this may be informed by the information used to construct the model. For example, a model based on the implementation of a system (using procedures, staff interviews, field observations etc.) is likely to feature more uses of 'required actor' and fewer of 'holds', as these sources focus on practical details. In contrast, a system based on higher-level sources (specifications, regulations, organisational structures etc.) may contain more 'holds' relations, and may lack 'required actor' relations where the details of how a specification or regulation should be met are not described.

Some responsibilities or resources may appear underspecified; for example, a resource that is not produced or a responsibility that has no required actors. This may occur for at least three different reasons. Firstly, the entity might be described with deliberately vague detail in order to avoid unnecessarily extending the scope of a system, such as a resource that is simply assumed to always be available to avoid showing the details of its production. Secondly, the entity might not have a clear and practical definition in the problem domain, such as fairly abstract legal requirements that organisations might hold, but where the details of how such obligations would be ensured in practice is unclear. Thirdly, it may simply be an oversight by the modeller.

5.5 Patterns

Responsibility models are models of socio-technical system structures. These structures may be formal or informal, and may or may not cross conventional organisational boundaries [42]. The complexity of socio-technical systems in breadth and scope may initially imply that there is little commonality between different systems, and that each responsibility model must be constructed and analysed independently from any others. However, common structures often appear across different domains and systems, and the recognition of these structures allows for effective model and system analysis.

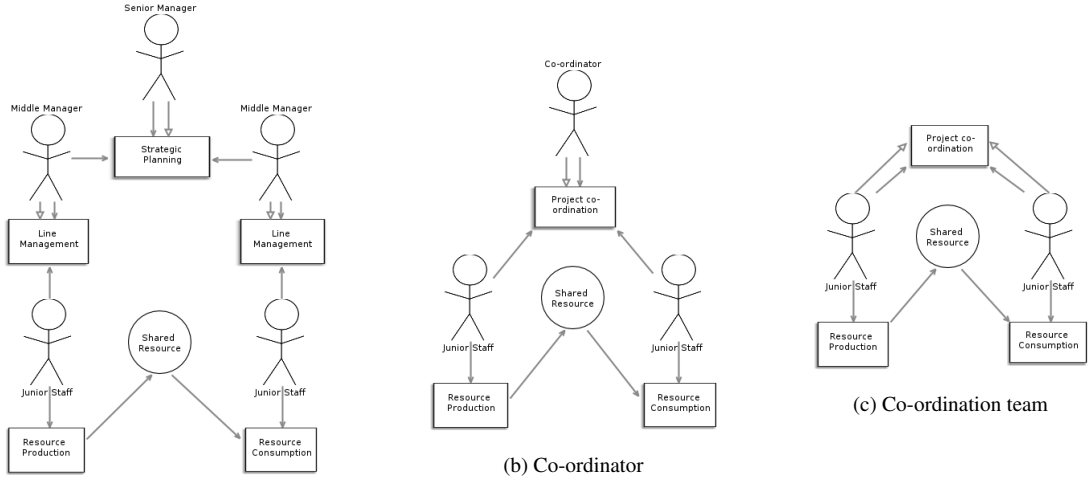
These patterns represent common structures in responsibility models - akin to design patterns in other forms of engineering. Software design patterns originated as a method for addressing common problems with a standard but adaptable solution to small-scale design problems [64]. Socio-technical design patterns have been studied in other contexts, such as Martin and Sommerville [121]'s ethnomethodologically-based patterns of work and Storer et al. [177]'s patterns of information security.

When used to analyse existing systems, patterns provide a library of structures that allow analysts to quickly identify features of the model and get insight to the vulnerabilities of that structure. When used for requirements engineering patterns provide simple structures with well-known strengths and weaknesses and guidance as to what contexts that structure is suitable or not suitable for.

This section defines and analyses common patterns for multi-agent co-ordination - a common activity in socio-technical systems [35], and a disproportionate source of potential vulnerabilities due to the common complexity and ambiguity of human communication, especially across organisational boundaries [170]. These definitions can be used to recognise and understand co-ordination within a system, and demonstrate the efficacy of the pattern analysis technique, which can be potentially extended to many other types of structure.

As models of organisational structures, responsibility models often feature implicit and explicit examples of multi-agent co-ordination. This co-ordination is required to meet some common goal, or to operate effectively as part of a (possibly virtual) organisation. While there are significant differences between different types of organisation at an abstract level there are very few distinct forms of co-ordination. Mintzberg [126] suggests that there are just five basic forms of co-ordination - ultimately, co-ordination is either ad-hoc at a flat level, or intermediated by some system or actor. Each abstract type of co-ordination has strengths and weaknesses, which lead to particular types of vulnerabilities in the actual system.

Explicit co-ordination generally takes the form of one of three patterns: a hierarchy, a co-ordinator or a co-ordination team. Both the hierarchy and the co-ordinator patterns are variations of Mintzberg's 'Direct Supervision', and the co-ordination team is equivalent to



(a) Hierarchy

(b) Co-ordinator

(c) Co-ordination team

Figure 5.6: Co-ordination patterns

'Mutual Adjustment'. Implicit co-ordination occurs when multiple actors are assigned to the same responsibility; potential vulnerabilities arise when co-ordination may be necessary but is not specified, such as between actors working in a long production-consumption chain. Mintzberg's three other co-ordination forms all represent types of implicit co-ordination. Figure 5.6 shows responsibility models of the three types of explicit co-ordination, using resource production/consumption as the shared work.

In the hierarchy pattern, actors are linked by the chain of actors above them - at some level these chains meet either by being supervised by the same superior actor, or by sharing a common responsibility. This structure provides weak co-ordination and loses power as it scales. Automatic analysis of this responsibility structure shows that dependency links are broken as low-level staff do not hold any responsibility for co-ordinating their activities; instead, senior management staff perform high-level co-ordination but rely on middle management below them, who are themselves reliant on the low-level staff. Middle managers are particularly at risk of overload if they manage large numbers of low-level actors. Additionally, this pattern creates critical actors that act as single points of failure - if a top-level actor fails large numbers of lower-placed actors will fail to co-ordinate. However, the hierarchy minimises load on lower-level actors by reducing their communication overload and also minimises the number of actor roles required.

In the co-ordinator pattern, actors with shared responsibilities communicate via an explicit third-party actor, the co-ordinator. The co-ordinator acts as a proxy, relaying communications between the other actors. When there are too few co-ordinators compared to other actors, this pattern shares the same strengths and weaknesses as hierarchy. When used more substantially, the co-ordinator pattern leads to a decentralised system that scales effectively and can retain robust communication. However, additional resources are required to provide the extra

co-ordinators, and communicating with multiple co-ordinators may overload actors. The co-ordination pattern places (potentially multiple layers of) co-ordinators directly above frontline staff, making each co-ordinator directly reliant on those actors. If co-ordinators hold responsibilities for excessive numbers of projects or staff then they face the same overload problem as middle managers in a hierarchy.

In the co-ordination team pattern actors with shared responsibilities do not use a separate co-ordinator and instead together form a co-ordination group with shared responsibility. In comparison to the co-ordinator pattern, this approach reduces the amount of actors required while retaining communication at the individual rather than the hierarchical level. It is counter-indicated when the co-ordination group would be excessively large, or where one actor would be part of multiple groups - in both cases it becomes impractical for effective communication to occur. In a co-ordination team actors communicate directly with each other without any intermediate layers. This increases the number of responsibilities held by each actor, potentially leading to overload. Additionally, this is the only pattern where co-operating actors directly rely on each, rather than having this link removed by intermediation.

A wide range of other pattern definitions are possible. For example, there are patterns apparent in resource management (assembly line, pooled resources, information alerts), responsibility delegation (matrix management structures, managerial hierarchies, team working) and inter-related responsibilities (check and audit, iterative processes, checklists etc.)

5.6 Actor Roles

In responsibility modelling actors represent roles, rather than actual individuals or organisations. One role might be fulfilled by several different organisations (an emergency first responder may be from the police, fire or ambulance services), while one organisation might fulfil multiple roles (a GP simultaneously fills the roles of medical advisor, NHS contractor and small business owner). This complexity in actor roles was recognised in Lock et al. [116], where the ‘Act As’ relationship was introduced to provide a link between individuals and roles (for example, ‘Bob acts as an administrator’). Unfortunately, neither a formal definition or an illustrative example were provided.

However, this approach risks muddying the definition of actors by introducing both roles and the agents performing the roles in the same model. This would require introducing more types of entities and relations, with a corresponding increase in model complexity. Given that role allocations are ultimately an implementation detail (the responsibility model is a general, abstracted view of the system, while role allocations may vary according to particular circumstances) it is best not to include them directly in the model. Alternatively, if directly

modelling actors is essential then it is possible to remove the roles and model entirely in terms of actual actors, accepting the reduction in generality that this entails.

Despite this, there are clear advantages to studying the relationships between actor roles and those fulfilling them. Conflicts of interest can arise if two seemingly separate roles are filled by the same actor - consider a regulator who also owns a business in their area of regulation. Simplistic versions of actor load analysis can also be subverted by actors holding multiple roles - two individual roles may each have manageable loads, but if performed by the same actor the combined load becomes excessive.

In addition, the ability of actors to fulfil multiple roles also impacts on criticality analysis and vulnerability detection. Individual roles may may not be critical to the system, but a combination of roles occupied by the same actor could be. Likewise, disabling all roles played by an actor may cause the system to fail, but disabling only one role will not.

A low-effort technique to study this is to introduce an additional, optional phase of actor analysis. In this phase, mappings are constructed between roles and actors, which indicate which actors fulfil which roles. In conjunction with the criticality and load values for the roles (produced by the toolkit) this can quickly be used to determine potential vulnerabilities in terms of actors. If it is necessary to examine the effectiveness of actors failing to discharge multiple roles then simply disable all roles they currently hold, and perform analysis as normal. By constructing a simple table of mappings between roles and actual actors it is possible to perform this analysis with relative ease and without relying on specific tool support. This is similar to Lock et al. [116]'s 'Act As' relationship, but separating the mapping of roles and actors from the model itself simplifies the semantics of the relationship.

When constructing the model, modellers can also place constraints on role allocation that should be checked during this process. For example, they can constrain the role of the regulator and the regulated person such that no individual can hold both roles. By explicitly identifying potential sources of vulnerabilities at the time of modelling it becomes extremely simple to detect them once the model is instantiated with actors. Again, this can provide useful warnings without the need to extend the responsibility notation and add additional complexity.

5.7 What-If Analysis

One use of socio-technical analysis is the evaluation of potential failures, and the effects that failures of individual components and subsystem have on the overall system. This type of evaluation can help to detect and correct vulnerabilities in a deployed system, or can be used to incorporate more redundancy into requirements or a system design. In many socio-technical methodologies this is obtained by performing a static analysis of the system

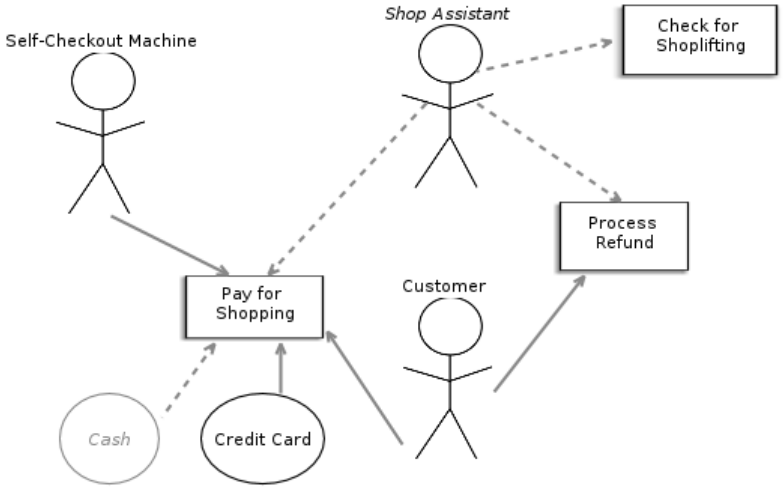


Figure 5.7: Responsibility model showing selectively disabled entities

(e.g. workability analysis in i* [202] or process analysis in STAMP [109]) - evaluating the structure and details of the model, and identifying single points of failure and potentially stressed elements; formalised responsibility modelling's automatic analysis techniques work in this way.

However, additional insight can be obtained by using dynamic analysis - modifying the model to represent particular scenarios or failure states, and then analysing how the model operates in these degraded conditions. This may involve re-applying static analysis techniques in the new situation, or evaluating the satisfaction of entities under the new conditions. To enable this form of 'what-if' analysis the RESME toolkit allows for the selective enabling and disabling of individual entities and relations; this has the effect of temporarily removing them from the semantic model while they remain visible (but clearly disabled) in the visual model. Model elements (both relations and entities) can be selectively enabled and disabled by double-clicking on them - disabling an element has the effect of removing it from the model for the purposes of analysis, which is visually indicated by greying-out the element.

Figure 5.7 shows a simple responsibility model of purchases in a supermarket. Two of the entities in this model have been disabled (as indicated by the italic text on the label, the grey border on the entities and the broken lines on their relations) - the resource Cash and the actor Shop Assistant. Visually, certain effects of these disabled entities are apparent - for example, the responsibility Check for Shoplifting cannot be discharged. However, more interesting results can be obtained by comparing the results of automated analysis run before and after the entities are disabled. In this example the most critical actor changes as a result of the disabled entities; with all entities enabled the most critical actor is the Shop Assistant with a score of three, while once disabled the most critical actor is now the Customer, with a score of two.

This process allows for ‘what-if’ analysis to be used much more widely than in notations where manual copies and variations of the model must be made for each change. As the responsibility model is formally defined and tool-supported, the state of the model updates immediately as each entity is disabled, and analysis can be quickly re-run to understand the implications of the change to the system.

‘What-if’ analysis can be performed with two main strategies. Firstly, the disabling of individual entities or small groups of entities allows for the simulation of the failure of specific subsystems or actors, and can be used to test the resilience of the system against small, individual failures. These can be combined into larger and larger sets of failures, allowing the system to be tested to destruction. Conversely, this can also be used to detect potentially over-engineered systems - if multiple sub components can be disabled without compromising the wider system then the level of redundancy provided may be overspecified.

Alternatively, ‘what-if’ analysis can be used to reproduce known failure cases or incidents. Initially, these can be used to calibrate and validate the model as an accurate representation of the actual domain. Running the model with the appropriate failed elements disabled should produce a result that is broadly similar to the results of the incident - a large difference between the simulated result and the actual result shows a lack of fidelity in the model. Once confidence has been established in the model it can then be used to predict the consequences of incidents, which is particularly useful when considering changes to alleviate future problems. Additional safeguards and preventative systems can be added to the model and the incident reconstructed, allowing the model to predict whether or not the changes will eliminate (or reduce the severity of) similar incidents in the future.

5.8 Conclusion

In this chapter we have presented a comprehensive package of analysis techniques for formalised responsibility modelling. By utilising the consistent definitions of entities and relations provided by formalised responsibility modelling we are able to define automated analysis techniques to quickly detect potential vulnerabilities in a model. Overloaded actors, resource production, points of failure and implicit organisational structures can all be detected automatically. Additionally, the use of satisfaction criteria allows for the detailed specification of the success requirements for individual entities, allowing model-wide evaluation of the success or failure of the system. A guide is provided to common model ‘problems’, which examines whether potentially concerning elements in models represent genuine problems with the model, or are in fact signs of unusual or incomplete definition in the socio-technical system itself.

Common traits in models also allow for the definition of patterns, which use responsibility

structures to provide insights into the strengths and weaknesses of the underlying system. Role analysis enables the investigation of the complex relationship between responsibility modelling actors and individual people or organisations, who may often act in more than one actor role. Structured and semi-formal techniques enable more insightful and probing analysis. 'What-if' analysis allows the simulation of individual entity failures to examine the impact on the operation and redundancy of the system. The majority of formal and semi-formal techniques are implemented in our RESME toolkit, which also provides easy-to-use graphical creation and editing of responsibility models.

Chapter 6

Case Study - A Re-Engineered Responsibility Modelling Study

6.1 Introduction

This chapter analyses a case study that was previously studied by conventional responsibility modelling. We have chosen the Hunterston emergency plan study, as previously analysed by the InDeED consortium [86] and validated against field exercises. The core of this study is the Hunterston Nuclear Power Stations Off-Site Emergency Plan [133]. This case study was chosen for two reasons: firstly, to evaluate whether formalised responsibility modelling is consistent with the original responsibility modelling approach; secondly, to investigate the advantages of the new approach by performing a more structured analysis, emphasising the value of tool support and the iterative and dynamic aspects of the analysis enabled by formal semantics.

By law in the UK, a local government authority area that contains a nuclear power station must produce an *Off-Site Emergency Plan* to collate the emergency arrangements of the various agencies that would respond to a radiation incident at the nuclear plants [185]. The Off-Site Emergency Plan for the Hunterston Nuclear Power Stations in North Ayrshire [133] is the subject of our analysis. The plan details the agencies involved, the communication between them and many of the actions taken in the surrounding area; it does not describe the actual process of emergency action at the site itself, which is the internal responsibility of the site owner.

By analysing this document, we show the utility of formalised responsibility modelling in a number of ways. These include:

- showing that responsibility models can be used to effectively represent complex scenarios (the plan runs to over 100 pages)

- showing that construction and analysis of responsibility models does not require domain-specific knowledge (we have no special knowledge of nuclear power or emergency planning)
- demonstrating that automatic analysis methods can provide insights as to the structure of the responsibilities in a system
- showing that our updated responsibility modelling is compatible with previous approaches, while offering useful new features

This scenario was previously studied by the InDeED Project, which developed the first forms of responsibility modelling. We have obtained copies of their models, observations and initial analysis. As a result, this study can be compared with theirs, which both validates our models and allows comparison between the two different techniques. The InDeED models were not externally validated or evaluated; the comparison allows for relative validation against the previous state-of-the-art but does not make claims about the inherent quality of the models and analysis.

Models produced and notes referenced during this study are available online in two GitHub repositories: [158] contains the responsibility models themselves (in the file format of the RESME toolkit), while [87] contains the model (in Visio format) and observational notes from InDeED's previous study.

The remainder of this chapter is laid out as follows. Section 6.2 provides the relevant background material for this study. This includes details on the Hunterston power complex and organisations involved in the emergency response, as well as discussing the scope and content of the Off-Site Emergency Plan itself. Section 6.3 presents the modelling process and its results. This includes an overview of the modelling procedure and comments on particularly interesting parts of the model, as well as challenges encountered during the process. Section 6.4 describes our analysis of the constructed model, based on both informal observation and automated analysis. Potential vulnerabilities in the emergency plan are examined, and the effectiveness of the approach is considered. Sections 6.5, 6.6 & 6.7 compare the results of our analysis with the earlier InDeED Project work, an updated version of the emergency plan and against observations obtained from emergency planning exercises. In Section 6.8 the dynamic features of the responsibility modelling toolkit are used to re-run elements of the field exercises and identify problems that arise. Finally, Section 6.9 provides a final overview and conclusion.

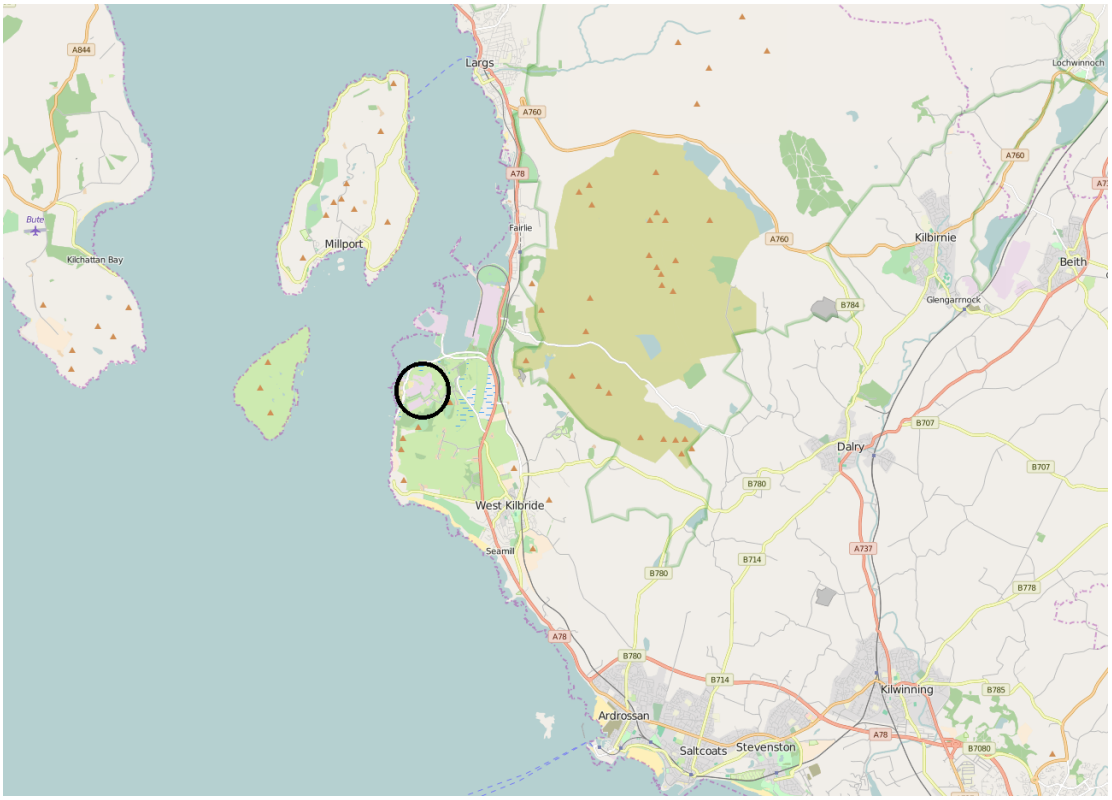


Figure 6.1: Map showing the location of the Hunterston site (circled) and surrounding communities
Map ©OpenStreetMap contributors, available under the Open Database License

6.2 Background

The Hunterston nuclear power complex consists of two separate facilities - Hunterston A, a now-closed Magnox nuclear power station and Hunterston B, a currently operational AGR (Advanced Gas Cooled Reactor) nuclear power station. Hunterston is located on the Ayrshire coast in western Scotland approximately 30 kilometres from the city of Glasgow, as illustrated in Figure 6.1.

The Hunterston power stations are of two distinct generations. Hunterston A was commissioned in 1964 and decommissioned in 1990. While no longer operational, decommissioning and maintenance work continues by British Nuclear Group. The adjacent Hunterston B was commissioned in 1976 and is currently (as of 2016) operational. It is operated by EDF Energy (formally British Energy Generation UK). Both sites contain hazardous nuclear material capable of causing significant harm to the surrounding population.

Hunterston is located just off the A78 road in North Ayrshire. The Clydeport coal terminal and Largs-Glasgow railway line are within 1km of the site to the north-east. To the south is the small town of West Kilbride, with the larger multi-town conurbation of Ardrossan, Saltcoats and Stevenson a short distance further. The town of Largs lies to the north.

The Off-Site Emergency Plan is a legally required document, produced by North Ayrshire Council on behalf of the Strathclyde Emergencies Coordinating Group (a umbrella group consisting of emergency services, NHS bodies, local authorities etc.). The exact meaning of the term 'off-site emergency' is not defined in this document, although several examples are given (significant release of airborne radioactivity, discharge of radioactive material likely to cause a hazard to the public). A similar document prepared for Wylfa power station by the Isle of Anglesey Council provides the clearer definition: 'A hazardous condition which results, or is likely to result in the need to consider urgent countermeasures to protect the public outside the site security fence from a radiological hazard' [88].

The document can broadly be divided into three parts. Firstly, the process for declaring an off-site emergency, and informing all relevant organisations. Secondly, the actions to be taken in the surrounding area during and immediately after the incident - evacuating local residents, activating roadblocks, informing the media etc. Thirdly, a description of the roles and responsibilities of the participating agencies (which includes much of what appears in the previous sections, but with some additions, deletions and inconsistencies). Much of the content consists of responsibilities that are broadly inter-agency in nature, such as co-ordination between the emergency services and the local authority, although the third section provides significant detail on specific, non-cooperative tasks. No detail is provided on emergency action on the site itself; only reactions and associated work in the surrounding area.

The particular version of the document used in this study is dated March 2006. There are two main reasons for using this version, rather than the most up-to-date. Firstly, our 2006 copy is significantly less redacted than the publicly available current version, and so allows us to perform a fuller analysis given the security restrictions on this type of material. Secondly, but more importantly, it is this version that was subject to analysis performed by the InDeED consortium, using an earlier version of responsibility modeling. We have performed our modeling separately, without consulting the original models until the analysis stage; we intend to show that our technique provides comparable analysis with the potential for more detailed evaluation. Additionally, it is possible to check any detected vulnerabilities with the current version, which may indicate the effectiveness of responsibility analysis compared to the regular document review process.

The roles and responsibilities section of the plan lists 16 different participating agencies. Some agencies are mentioned in the process-based sections of the plan, but not listed in the set of agencies (a potential vulnerability that will be examined in Section 6.4). The main stakeholders are the site operators (British Nuclear Group, British Energy Generation UK), emergency services (NHS Ayrshire & Arran, Strathclyde Police, Scottish Ambulance Service) and government authorities (North Ayrshire Council, Scottish Executive, HM Nuclear Installations Inspectorate).

6.3 Initial Modelling

In order to model a scenario of this size subdivision of the problem is required. Firstly the sections of the emergency plan referring to specific sets of responsibilities were modelled. In practice, these were split by chapter, as each chapter of the report broadly describes a self-contained set of responsibilities. The details of each chapter were used to produce submodels describing these activities. Generally the content of these models corresponds directly to the chapter title, with the occasional exception (such as Chapter 7, 'Key Locations' corresponding to a model describing the establishment of various field sites). These exceptions generally arose when a chapter contained both information for reference and planned action - the title would correspond to the reference information, but it is the planned action that can be modelled.

For clarity and brevity, certain trivial responsibilities have been omitted such as 'Send representative to NAECC' (the emergency co-ordination centre), and such agents will not appear if they are only planned to attend, and not to perform any substantive action. Certain abstract responsibilities (for example, 'discharge legal requirements') have also been omitted, as they are too imprecise to analyse. Likewise, 'standard responsibilities' have been omitted if they restate normal day-to-day activity (for example, the NHS running hospitals). These responsibilities are unlikely to play an important role in the discharge of the system, and their omission simplifies the modelling and analysis process.

Additionally, it is important to emphasise that the model focuses on representing the off-site aspects of the emergency; the Off-Site Emergency Plan occasionally features aspects of on-site emergency planning, which have not been included. This effectively excludes certain agencies. For example, the Fire Service plays a vital role in dealing with the emergency on-site, but is described as having only advisory duties with regards to the off-site aspects.

Within each chapter, we first aimed to identify the relevant responsibilities. These were generally actions to be performed or processes to be followed. We have not attempted to model more abstract responsibilities (public duties, legal requirements) as the model concerns responsibilities described in this document, which generally contains specific contextual actions rather than overarching duties. Having identified responsibilities, corresponding actors and resources were determined. In most cases this was straightforward, as the clear language of the plan contained the required information. The main issue was identifying the difference between responsibilities that required an organisation in general (e.g. Strathclyde Police) and responsibilities that required a specific part of an organisation (e.g. the 'Incident Commander', who in most circumstances is a senior officer of Strathclyde Police). The division in chapters led to submodels being of a manageable size (4-10 responsibilities). A typical submodel is shown in Figure 6.2. This submodel is accessible as 'Rest Centre.responsibilitymetamodel' in [158], as are all other submodels.

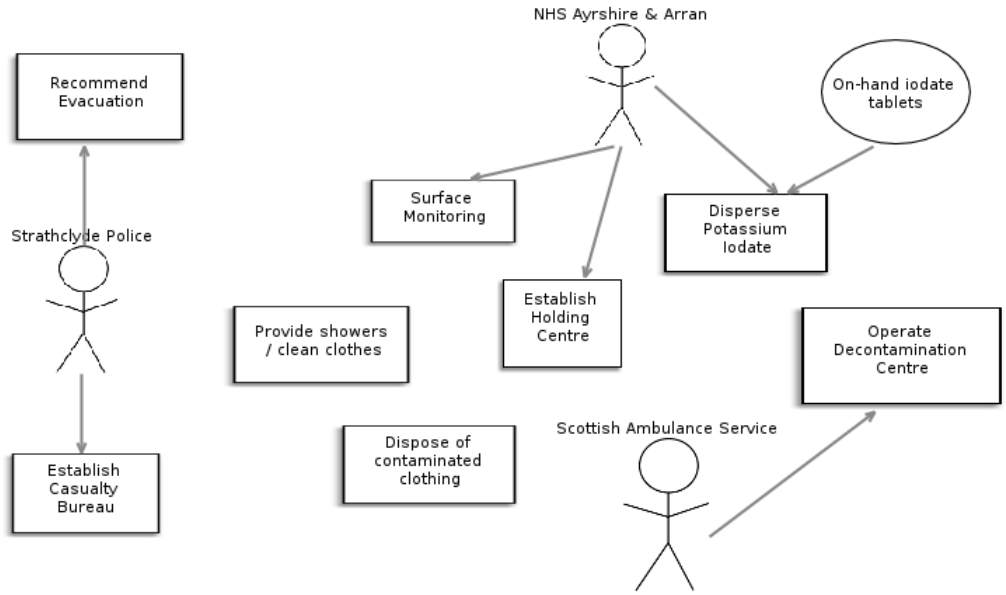


Figure 6.2: Sub-model showing responsibilities for Rest Centre management

This model shows the establishment of Rest Centres to accommodate those displaced by the emergency. Responsibilities are divided between different agencies - the Police are responsible for initiating the evacuation and tracking casualties, the Ambulance Service are responsible for decontamination of evacuees and the NHS are responsible for operating the centres and treating evacuees. Some responsibilities are explicitly defined in the plan but not assigned to any particular agency - responsibilities for handling evacuees who are forced to discard their clothing due to contamination. Only one responsibility requires resources that are explicitly mentioned - the iodate tablets for preventing thyroid cancer.

The process chapters of the emergency plan identify most of the responsibilities to be modelled. However, they are not the only relevant sections of the plan. Chapter 14 of the plan provides a comprehensive list of agencies and their agreed actions and responsibilities. This includes a combination of more abstract responsibilities and duties, as well as restatement of actions already covered in the previous chapters. We chose to model this section as one large model, which included all the listed stakeholders and the actions they had agreed to. After creating this model, it was combined with the individual subject-specific sub-models created earlier. This process was straightforward as there were very few direct inconsistencies between the two different views of the scenario, although many responsibilities described in the final section were not present in the individual sections, and vice versa. The combined model is a substantial construction, containing over fifty entities, as shown in Figure 6.3. This full model is accessible as 'Combined.responsibilitymetamodel' in [158].

The standard responsibility modelling notation presented in Chapter 4 was sufficient to capture the vast majority of the scenario. However, extensions were required to model

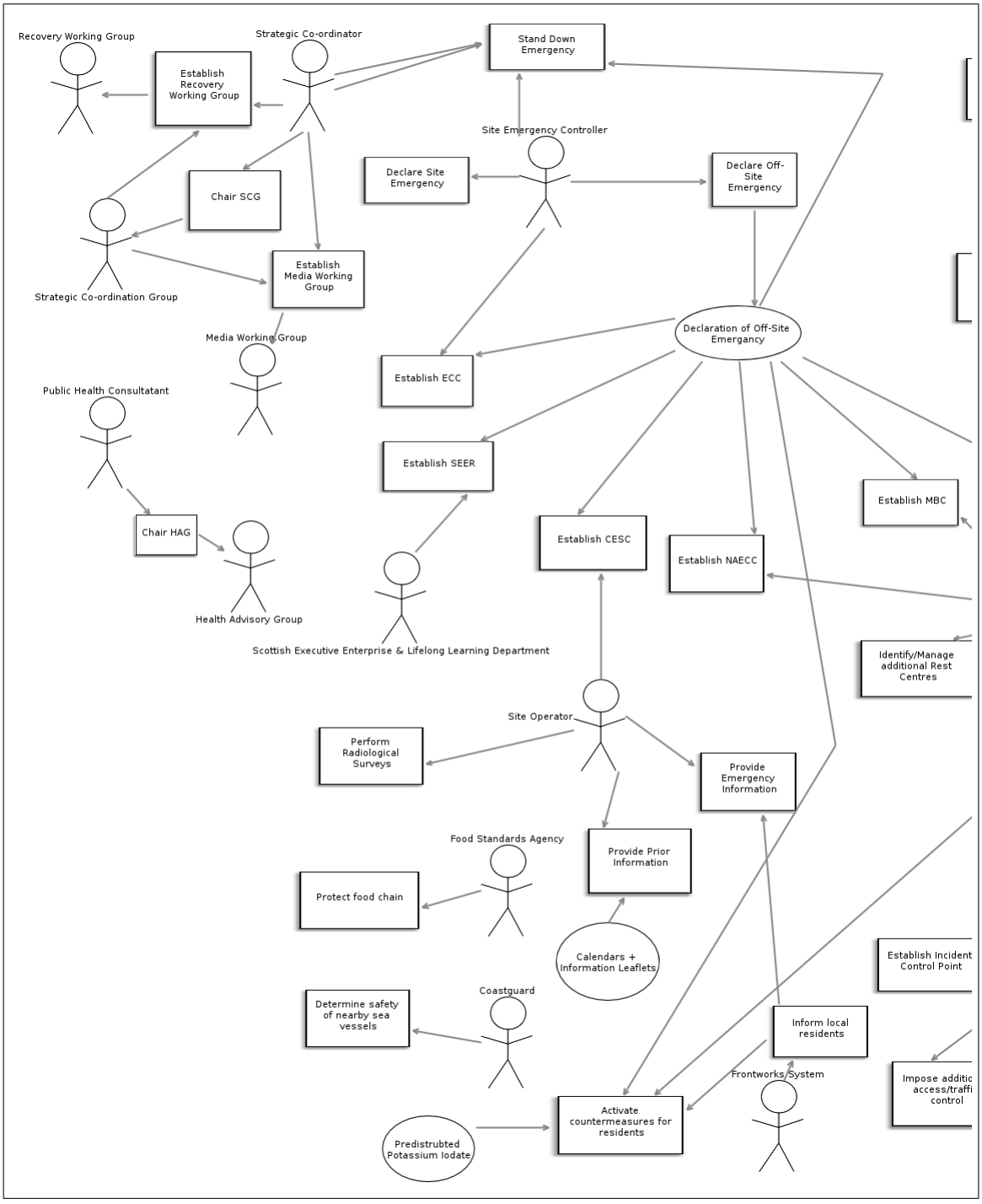
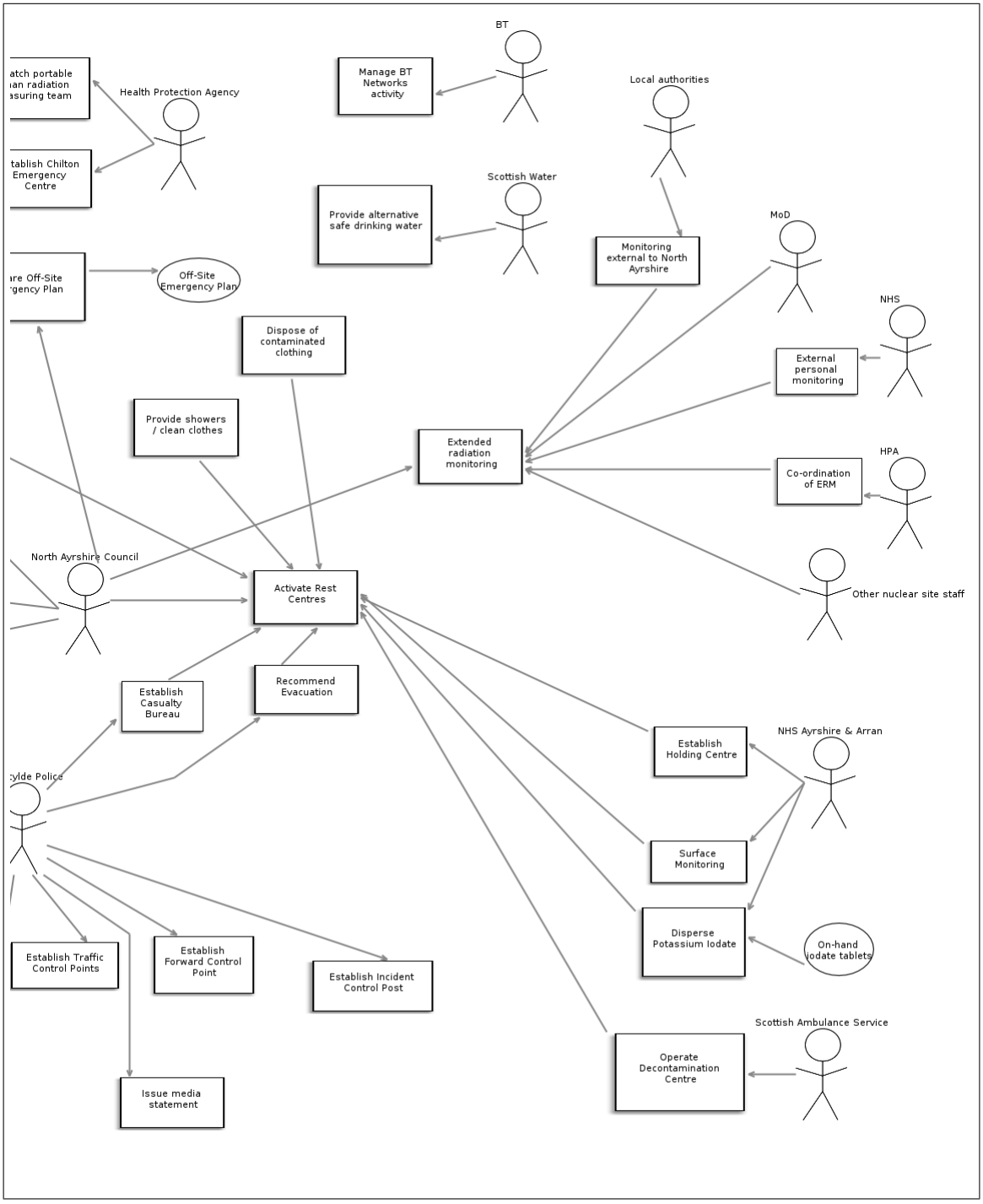


Figure 6.3: Completed responsibility model of the Hunterston emergency plan


BT
Manage BT Networks activity
Local authorities
Health Protection Agency
atch portable ian radiation asuring team
tablish Chilton Emergency Centre
Scottish Water
Provide alternative safe drinking water
Monitoring external to North Ayrshire
MoD
NHS
External personal monitoring
re Off-Site gency Plan
Off-Site Emergency Plan
Dispose of contaminated clothing
Provide showers / clean clothes
Extended radiation monitoring
HPA
Co-ordination of ERM
North Ayrshire Council
Activate Rest Centres
Other nuclear site staff
Establish Casualty Bureau
Recommend Evacuation
ylde Police
Establish Holding Centre
NHS Ayrshire & Arran
Surface Monitoring
Disperse Potassium Iodate
On-hand iodate tablets
Establish Traffic Control Points
Establish Forward Control Point
Establish Incident Control Post
Scottish Ambulance Service
Operate Decontamination Centre
Issue media statement

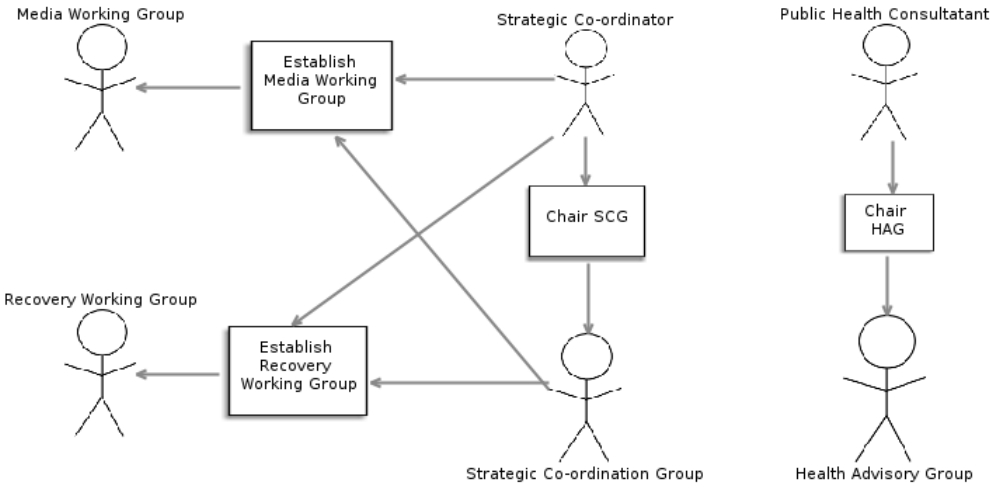


Figure 6.4: Sub-model showing establishment of working groups

the establishment of working groups (as part of establishing the Emergency Co-ordination Centre) in an intuitive way. In this part of the plan various committees and working groups (Media Working Group, Strategic Co-ordination Group) are established, and then take on responsibility for certain actions.

In the core responsibility modelling notation, only resources can be produced by responsibilities. We could therefore model these committees as resources - they are created by a responsibility 'Establish working group', and the tasks they perform can be stated to require these resources. Unfortunately, this approach is inelegant in two ways. Firstly, it is unintuitive - a committee is not a resource, it is an organisation - ease of understanding and reasoning by non-expert users is one of responsibility modelling's strengths, and over-use of this pattern would jeopardise this. Secondly, this type of mis-classification undermines the effectiveness of analysis. For example, reliance analysis will fail to include committees if they are modelled as resources. Likewise, attempting to determine implicit supervisory relationships will be affected and any actors sitting below the committees will be ignored.

Instead, we extended the notation by introducing a 'Produce Actor' relationship between responsibilities and actors, as shown in Figure 6.4. This is essentially the same as 'Produce Resource' but with a different output type. As a result, actors are no longer inherently satisfied without conditions; actors can be specified as being produced in the same way that resources are. Actors produced in this way are available and active if a responsibility that produces them is successfully discharged; actors without any producing responsibilities are assumed by default to be available, subject to any specified satisfaction criteria. Both graphical and textual representations are inherited from 'Produce Resource' with the relevant types exchanged.

6.4 Analysis

This section presents the analysis of the Hunterston Off-Site Emergency Plan responsibility model. Firstly, analysis is carried out by inspection and non-formalised evaluation of the model, as well as observations drawn from the process of modelling itself. Secondly, the model is analysed using the automatic analysis techniques of the RESME toolkit, which provides five different forms of structured analysis.

6.4.1 Model Inspection

The modelling process itself reveals potential vulnerabilities in the plan, even without the need to perform model analysis. The initial process chapters of the plan contain organisational actors that are not then included in the list of stakeholders. Most notably, the Ministry of Defence (MoD) is expected to participate in the extended radiation monitoring (Chapter 9.8). There are several MoD sites within travel distance of Hunterston, and the military can be expected to have the required training and equipment. However, neither the MoD or any part of the armed forces are listed as having agreed actions. They are not on the distribution list for the emergency plan. No mention is made in the plan about how the MoD will be contacted in an emergency, or which agency will be responsible for calling them in. As a result, there is a risk that the MoD will not respond when the plan calls for their support. Likewise, extended radiation monitoring calls for personal monitoring of individuals that have left the immediate area. However, individuals leaving the immediate area are likely to leave the area of NHS Ayrshire & Arran, and other health boards are left out of the plan in the same way as the MoD. Neighbouring local councils are called on to provide services and the other Ayrshire councils to the south are included, but councils to the north and east are ignored.

Reviewing the model it becomes apparent that there are certain core actors (site operator, NHS, North Ayrshire Council etc.) and a good number of actors that have only a peripheral involvement (Coastguard, British Telecom, Food Standards Agency). These core actors are linked to each other by shared responsibilities, required responsibilities or resource interactions, while the peripheral actors have responsibilities that stand alone. This is indicative of a lack of consistency in the plan, where responsibilities have been added due to legal or organisational obligations rather than an inherent functional need or priority. However, this does not mean that all peripheral responsibilities are oblique or unnecessary. The provision of uncontaminated drinking water is clearly important (but is not integrated with other aspects of the evacuation process). The Coastguard certainly have a role to play, especially given there is a major coal unloading terminal within two miles of the site. In contrast, British Telecom's responsibility to manage their network effectively does not seem to be a priority, or to require them to act differently from a non-emergency situation.

The Off-Site Emergency Plan is an overview and co-ordination document, rather than a detailed guide to implementing emergency procedures - such implementation details are more likely to managed in the individual response plans for each responding agency. While some inconsistencies were identified in the involvement of agencies by examining those missing from the distribution list, it is likely there are more cases where the response outlined in the emergency plan does not correspond with the agency's own plans. However, identifying and analysing these inconsistencies would require access to the internal operational plans of emergency services like the police and fire brigade, as well as government agencies such as the Nuclear Installations Inspectorate and the Scottish Executive's emergency response team; for understandable reasons these detailed documents are not in the public domain. Without access to these documents it is assumed that the response indicated by the emergency plan corresponds directly to the planned response by the agency, although there is reason to believe that undetected inconsistencies remain. As a result, analysis of the emergency plan alone cannot reveal all potential vulnerabilities in the emergency response, but instead reveals vulnerabilities within the emergency plan itself.

6.4.2 Automatic Analysis

The previous section provided an informal analysis, informed by the process of creating a responsibility model, as was undertaken in the InDeED project. The strength of formalising responsibility modelling is that analysis can be applied automatically by the toolkit, reducing analyst effort and producing consistent results. In this case study, five relevant analysis methods were applied: criticality analysis, overload detection, resource evaluation, assignment checks and reliance. The results of each type of analysis are discussed separately below.

Production & Assignment

Resource production analysis checks to see if any resources are featured in the model without being produced by a corresponding responsibility; similarly, assignment analysis identifies responsibilities that are not held by any actors. In both cases these checks identify either parts of the model that have not been specified in sufficient detail, or detect genuine oversights in the actual system.

Analysis of resource production reveals three cases of resources being consumed without previously being produced. These are two cases of potassium iodate tablets (a useful counter-measure to prevent thyroid cancer) not being made available and no production of emergency preparation documents (sent to residents and to be referred to in an emergency). However, these may not directly represent a vulnerability. The processes and responsibility for creating these resources do not appear in the model because they do not appear in the plan; this is

presumably because they are out-of-scope, as their creation occurs well before the emergency actually happens. This is not necessarily a vulnerability of the emergency plan, but instead amounts to a set of preconditions that are assumed to be in place.

In this scenario, regular checks or audits of these resources would be useful, to ensure that they will be available if necessary. This may actually happen already, but the plan does not mention them. This lack of detail in planning can lead to failures when assumed resources are not made available. For example, Sommerville et al. [170] report that several emergency agencies required evacuation information, but no agency held the responsibility for collecting it; as a result, this information was not available when an emergency occurred. In general, this can be seen as a weakness of the plan that contains only actions that occur during the emergency itself; the plan could be strengthened either by including a set of required preconditions (resources, organisational links etc.) or by including validation and review processes that occur regularly despite the absence of emergency.

Two warnings are generated for responsibilities that are not held by any actors. These two responsibilities are both part of the Rest Centre management plan - the disposal of contaminated clothing and the provision of showers and replacement clothes. Neither has an actor assigned. In some cases, it could be assumed that these responsibilities would be discharged by the actor assigned to their top-level responsibility: in this case, North Ayrshire Council, who hold the 'Activate Rest Centres' responsibility. The breakdown into sub-responsibilities would therefore be for clarity, or to more accurately model resource consumption. However, in this particular scenario that is not the case: the other sub-responsibilities of Rest Centre Management contain explicit assignments to actors within the emergency plan and these two responsibilities are the only ones that are not specified in this manner. In an emergency scenario it is entirely possible that no organisation will consider providing these facilities their responsibility, and hence it will not be possible to deal with evacuees that have been exposed to surface contamination.

Criticality Analysis

Criticality analysis identifies the entities (one each of actors, resources and responsibilities) that are most often relied upon to discharge the system's responsibilities - the entities that would cause the highest number of consequential failures if they failed themselves. This highlights the most vulnerable single points of failure in a system, and can be used to check if adequate safeguards or fallback mechanisms are in place.

In this case, criticality analysis provides a clear result. The most critical actor, resource and responsibility are all part of the chain for declaring an offsite emergency - 'Site Emergency Controller', 'Declaration of Off-Site Emergency' and 'Declare Off-Site Emergency' respectively. This might be considered obvious in the analysis of an emergency plan - all other

reactions to a emergency are not initiated until the emergency actually occurs. However, this may lead to a failure in particularly catastrophic circumstances. Only the site controller (or their deputy) can declare an off-site emergency, and the emergency plan is specifically built around reactions to such a declaration. If the site controller is incapacitated as a result of the initial incident there is no mechanism documented in the plan for an off-site emergency to be declared. Although it is reasonable to expect that first responses to the incident would still occur by the appropriate bodies, it would likely be ad-hoc response until a command structure is established.

Disabling the emergency declaration chain and re-running criticality analysis leads to a wider spread of critical entities, but also reveals some limitations in the analysis. The most critical actor is Strathclyde Police, who hold more than half a dozen distinct responsibilities. This is very clearly correct - the police are responsible for important tasks such as securing the site, considering evacuation and handling casualties. There are multiple candidates for the most critical responsibility and most critical resource that share the same criticality score, and so it is not possible to identify the single most significant point of failure. The model is relatively flat, with few complex interactions between entities. This represents the modular nature of the emergency plan, which is defined in chapters with minimal cross-links. The analysis therefore indicates the decentralised and federated nature of the system, rather than pinpointing specific vulnerabilities.

Overload Analysis

Overload analysis identifies actors holding large numbers of responsibilities, which may indicate that they are at risk of impaired performance due to the difficulty of satisfying multiple responsibilities simultaneously. Overload detection (using a threshold of three responsibilities) shows that five actors in this model are overloaded. Three of these are organisational actors, while two are individuals. Overloaded individuals are of particular concern, as individuals may be less capable of discharging multiple concurrent responsibilities.

The first overloaded individual is the Site Emergency Controller. In this case vulnerability is unlikely, as their responsibilities are temporally distinct. The controller is responsible for declaring an emergency (possibly on- or off-site), jointly responsible for establishing the emergency control centre and jointly responsible for standing down from the emergency state. In practice, these responsibilities are discharged consecutively so they are never responsible for more than two active responsibilities at the same time. The other overloaded individual actor is the Strategic Co-ordinator, who holds a number of responsibilities to establish and chair working groups. These responsibilities all require action at broadly the same time so there is a clear risk of delays or mistakes due to overload. However, the co-ordinator is likely to be supported by other staff in the emergency co-ordination centre, so the risk is less

substantial than an individual operating entirely alone.

Of the three overloaded organisational actors Strathclyde Police are the most overloaded, holding nine different responsibilities. The main focus of their responsibilities is securing the area and establishing control points, and their secondary focus is managing the evacuation of civilians. As a large police force this range of responsibilities is understandable, and should be within the ability of the force to cope. However, it is important that they effectively balance their resources and deploy themselves proportionality to the right responsibilities; impaired performance could occur if they over-prioritise some responsibilities while under-prioritising others.

North Ayrshire Council is the second most overloaded actor. Their main responsibilities are activating rest centres for evacuees as well as establishing the emergency co-ordination centre. This provides a useful geographic and functional split between the organisational work of their headquarters staff in establishing the co-ordination center and the practical, distributed work in the rest centres. As a result this overload should not be a problem as long as it is properly prepared for. Finally, the site operator is overloaded. As with the site controller (themselves part of the site operator organisation) their tasks are temporally distributed - including some responsibilities ('Provide Prior Information') that occur before any emergency has been declared. Given this, serious vulnerabilities are unlikely to occur.

Reliance

Reliance analysis generates lists of actors that rely on other actors in order to discharge their own responsibilities. This shows the interdependencies between actors, while relying on actors in different organisations highlights potential vulnerabilities caused by lack of control. Reliance analysis on the model highlights the high interconnectedness of the system. No individual actor stands out as either strongly independent or highly dependent; all the main actors (those holding multiple responsibilities) demonstrate substantial reliance on others. This is to be expected - the Off-Site Emergency Plan is a multi-agency planning and co-ordination document, so a significant focus is given to interactions between actors.

It is clear that the emergency plan can only function as a combined effort of the relevant bodies; it is not possible for any particular organisation to heroically substitute itself for others. The set of actors not relied upon by any other is essentially identical to the 'peripheral' actors discovered by visual inspection, providing a more methodological definition of the same concept.

6.5 Comparison with Previous Modelling Effort

The InDeED consortium previously modelled the Hunterston Off-Site Emergency Plan using an earlier form of responsibility modelling. As part of a wider research investigation into responsibility and trust in socio-technical systems, responsibility analysis was carried out on several domains broadly focused around infrastructure - voting procedures [115], emergency flood planning [170] and nuclear safety planning [174].

However, the Hunterston study differs from the other InDeED responsibility studies as it was never formally written up and published; a full model was produced with some accompanying notes, but it never progressed to a working paper. It should therefore be taken with certain caveats as a non-published, non-peer reviewed source. Additionally, the evaluation performed on the InDeED model is own our, rather than InDeED's; only the model itself was produced by InDeED and any errors or omissions in the discussion are our own.

Their study consists of 26 different sub-models, covering both the organisational and functional elements of the plan [174]. These sub-models are broadly comparable to the sub-models produced during the development of our model, but elaborated to contain more detail. For example, Figure 6.5 shows the InDeED model for responsibilities involving the Coastguard. Our model (Figure 6.3) contains only one (abstract) responsibility involving the Coastguard, while the InDeED model also contains a breakdown of different nautical communication methods and a taxonomy of different types of shipping. This information is all contained in the relevant Coastguard section in the emergency plan; InDeED chose to model the plan effectively unabridged, while we simplified for ease of modelling and analysis.

This trend holds throughout the InDeED model. Sections from the emergency plan are fully modelled, containing all responsibilities listed. Organisations are often decomposed into sections or individual roles (for example, North Ayrshire Council is decomposed into Social Work Department, Chief Executive, Communications Officer etc.). Approximately one third of the model consists of these decompositions with the remainder consisting of responsibility assignments. At no point are the sub-models explicitly linked; conceptually they could be linked by identifying common elements (the same actors and responsibilities appear across multiple sub-models) but the modelling package used by InDeED (Microsoft Visio) does not support this directly.

In comparison to InDeED, our model is smaller and more compact, but lacks detail in some areas. This is not fundamental to formalised responsibility modelling or the construction methodology; this level of abstraction seemed most efficient for obtaining useful results, but we could model to the same level as InDeED as well. This would require a increase in modelling time as our approach is to model in sections, and each section must be manually linked together; the time taking for linking quickly increases as the number of sections

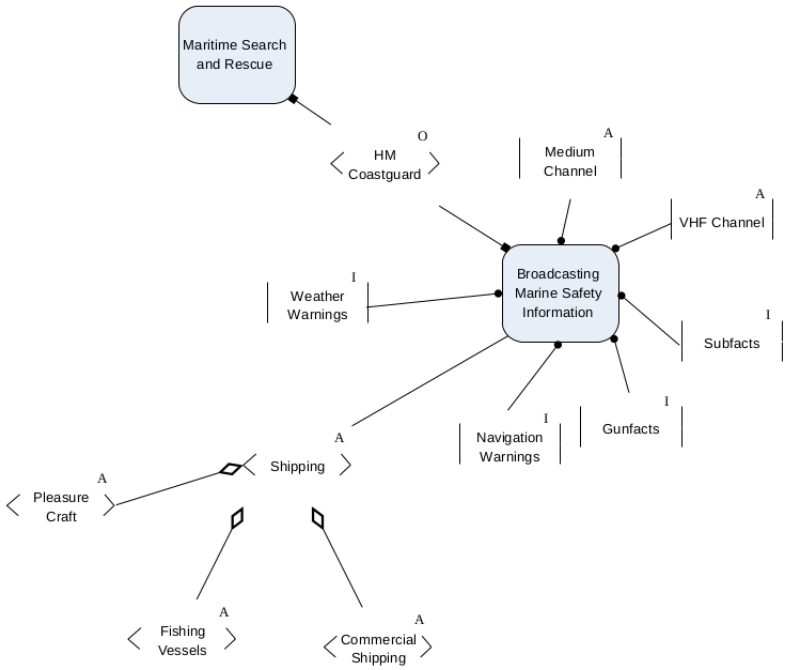


Figure 6.5: InDeED sub-model representing Coastguard and activities

increases.

On the contrary, formalised responsibility modelling does not have an easy way of expressing the actor decomposition that InDeED has. The concept of instance layers allows the mapping between abstract roles and specific organisations or individuals, but does not allow the modelling of situations where one actor is divided into subcomponents. Responsibility decomposition is supported, and this can be used to create implied actor decomposition by assigning sub-responsibilities to clearly labelled actors. While this slightly limits the coverage of the modelling it is this relatively simple set of relationships that allows effective automatic analysis, which is the key improvement over previous versions of the methodology.

6.6 Comparison with Updated Documentation

Having performed our analysis on the same 2006 plan as InDeED it is possible to compare issues raised by the analysis to changes made in the plan since, with the latest version being issued in June 2015 [7, pgs. 5, 8, 65]. (This version is subject to more substantial redaction than the 2006 version, particularly around site and operational locations). The plan has been extended and sections expanded, with additional detail on facilities in the evacuation zone, post-accident recovery and related documents. Additionally, a number of participating agencies have changed name and structure as a result of the Scottish Government's merger of emergency services.

Some of the issues raised by our analysis have since been detected and addressed during the annual revisions of the plan. The distribution list has been updated to include a wider range of neighbouring local authorities and health boards, and the previously required but unincluded Ministry of Defence has been removed from the plan entirely. However, most of the issues identified remain relevant.

The Site Emergency Controller remains the only entity able to declare an off-site emergency and initiate the plan; there is still no secondary measure to account for their unavailability. Details around the operation of rest centres remain unclear, but the duties of the relevant departments of North Ayrshire Council have been expanded and managerial (if not operational) authority clearly rests with the social work department. Communication between agencies has been improved by specifying some additional details (such as more information on communications equipment within the emergency centre) but the general process of inter-agency communication is still not documented.

6.7 Emergency Exercises

InDeED attended two emergency planning exercises testing the plan. 'Exercise Kilchattan' tested reactions to a waste spillage from Hunterston A [173], while 'Exercise Indus' reacted to a fire and gas leak from Hunterston B [172]. Notes from Exercise Kilchattan mainly critiqued the running of the exercise itself, but noted some more general issues. The number of organisations involved meant that discussion were sidetracked by the interests of peripheral agencies - the Coastguard were identified as being particularly keen to contribute despite not being relevant to the vast majority of tasks (the peripheral nature of the coastguard and similar agencies was identified in Section 6.4). The exercise was also impaired by communications problems - the phone lines at the Nuclear Installations Inspectorate were down, and the plan does not contain fallback procedures to address this. Geographical factors also impaired the smooth running of the emergency response - the Central Emergency Support Centre (CESC) would normally deploy a team to the site as the situation developed, but this is not practical given the location of CESC in southern England. As a result, communications problems emerged given the need to co-ordinate between CESC and local staff.

Notes from Exercise Indus also emphasised problems with communication. Communications between British Energy (the site operator) and the emergency control room were sporadic and incomplete, leading to confusion about casualty details and the extent of the radioactive gas leak. Communications between the different responding agencies were also poor. Most were co-located in the same control room and had deployed with their own computers and recording equipment. However, no provisions were made for exchanging information electronically between agencies so messages were passed by using double-copy carbon paper. Some use was

made of shared artifacts such as a whiteboard used for status updates but this was sporadically updated and so often inaccurate.

Problems also arose with some of the on-ground activities. Restrictions on traffic movement were imposed too slowly, allowing a passenger train to move through the area of radioactive gas. Traffic control and operational points were moved as the plume of gas spread, but this was recorded only in the police's STORM command & control system and not adequately expressed to other agencies. As one employee was said to have 'died' during the incident the site of the power plant was declared a crime scene, which led to conflicts between the police's responsibility to secure the crime scene and their responsibility to stabilise the site and effect repairs.

A number of plan areas were not tested during these exercises - for example, neither medical treatment or rest centres were covered. As a result it was not possible for some of the problems identified by our analysis to occur. No actors were incapacitated during the exercises, so criticality problems did not arise. There was some evidence of overload problems, especially amongst the police (identified as the most vulnerable actor) exacerbated by communication issues.

In general, communication issues emerged as the most serious problem and led to some impaired operation. Modelling communications mechanisms and resources in more detail in the responsibility model therefore appears to be a useful way to identify additional issues. Unfortunately while the emergency plan is clear on what is to be communicated it lacks detail on the communications channels to be used. General knowledge and comments from the field exercises suggest a mix of medium-range radio and landline telephones are used for communication with the field, while a memo system is used to pass messages within the control room. Neither of these are documented within the plan and may not hold true in the future; additionally, there are number of special purpose communication links in use, such as the TiiMs system for communicating with British Energy.

6.8 Simulation of Exercises

Our analysis identified limitations in both the responsibility model and the emergency plan itself. It is clear that the model is not a completely accurate representation of either the plan or the underlying emergency response. This is entirely to be expected, as the plan itself does not fully or comprehensively reflect the reality of the emergency situation, and a semi-formal model cannot be expected to capture the entirety of a free-text document.

However, the model can be modified to more accurately reflect the underlying scenario. This requires deviating from the strict structure laid down by the emergency plan and redesigning the model to more closely follow operational realities. Non-core and non-immediate (not

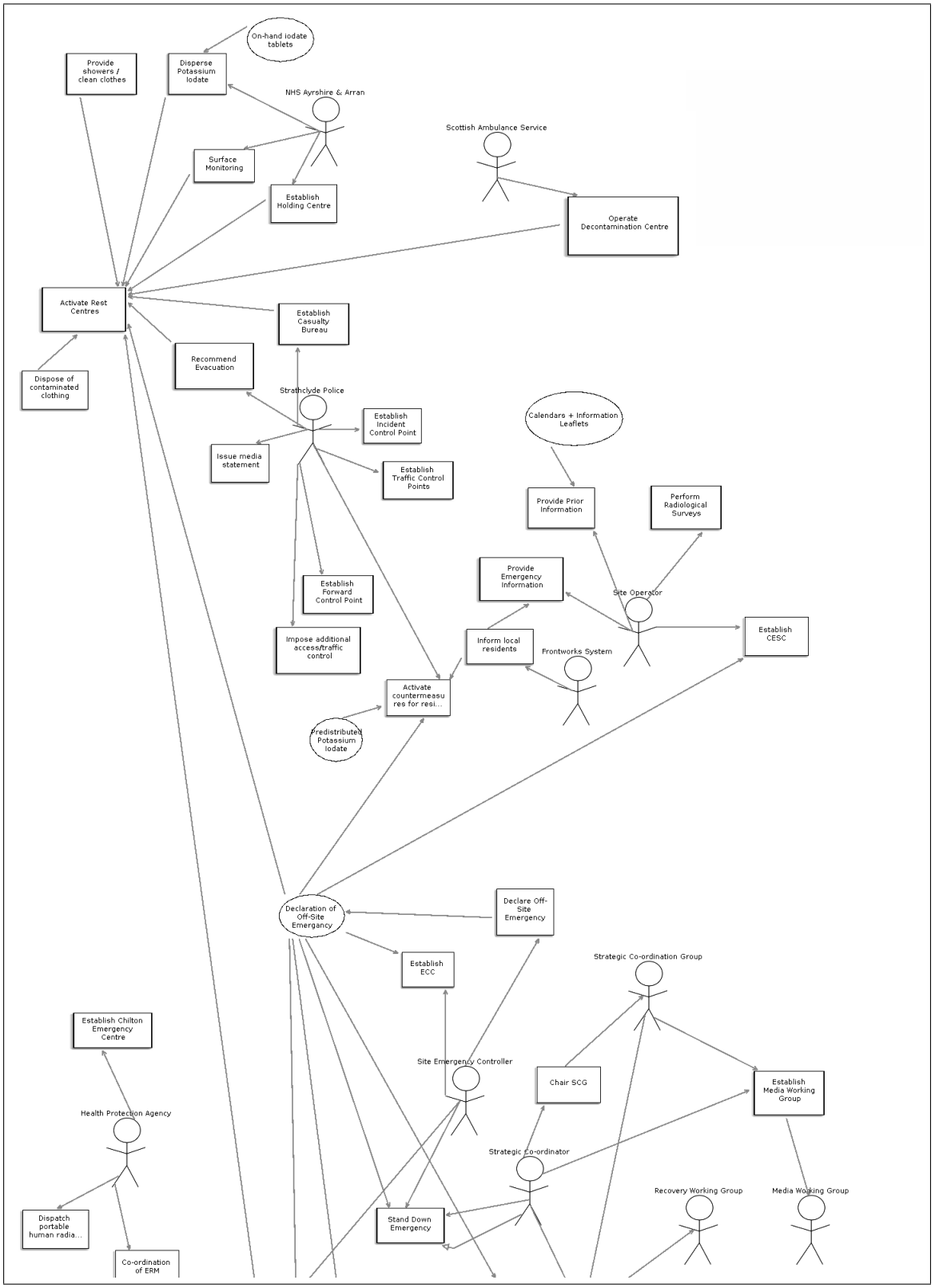
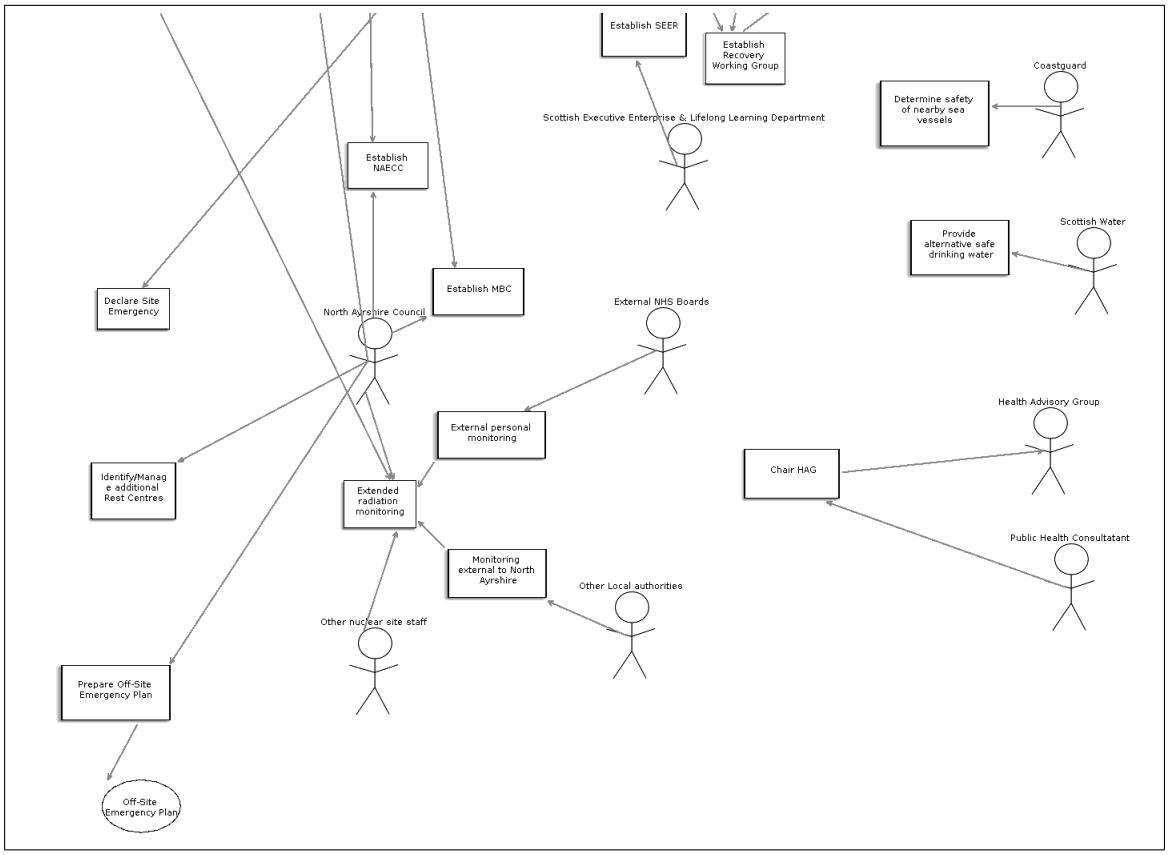


Figure 6.6: Revised responsibility model of the Hunterston emergency plan



relating to the period directly following the declaration of an off-site emergency) actors and responsibilities were removed. These included the removal of BT's management of communications networks and the Food Standards Agency's protection of the food chain; the first is is non-core and difficult to clearly specify, while the second is important but unlikely to begin until the situation has been brought under general control. The model was also updated with relevant information from the revised public version of the emergency plan - deleted actors like the MoD were removed, and details on previously ambiguous assignments such as the roles of various NHS bodies clarified. However, the structure and names of agencies were kept at their 2006 versions, rather than updating to reflect changes in nomenclature (such as the establishment of a national police force). These updates align the model more closely to the situation 'on the ground' in 2006, as described in the plan and examined by field exercises. This revised model shown in Figure 6.6 and is accessible as 'Hunterston2.responsibilitymetamodel' in [158].

Now that the model has been updated to reflect (with a reasonable degree of fidelity) the operational situation at the time of the emergency plan it can be used to simulate similar scenarios to the planning exercises observed by InDeED. The responsibility modelling toolkit offers the ability to disable selected elements of a model, and to then perform analysis indicating the effects of these changes on the wider system. This can be used to simulate an accident that triggers the emergency plan and knocks out certain parts of the emergency response, and

indicates the effectiveness of the response under these constrained circumstances. In effect, partially automated paper or tabletop exercises can be performed that test the emergency plan and suggest areas for improvement, using considerably fewer resources than full-scale multi-agency exercises. This type of analysis is not a direct substitute for full-scale exercises, but may allow some issues to be identified earlier and at lower cost.

6.8.1 Exercise Indus

Exercise Indus simulated an incident involving the release of airborne radioactive materials from Hunterston B, as well as an on-site fire. Specific constraints occurring in this exercise included the failure of the TiiMs computerised information system and substantial delays in establishing contact with British Energy staff on-site. To simulate these issues, the Site Operator actor in our model was disabled (reflecting the inability to reach either site staff or automated information). The Site Emergency Controller was left activated despite being an on-site member of British Energy staff, as according to the exercise documentation the site controller was successfully able to declare an off-site emergency.

With these constraints simulated, analysis of the model reveals a number of failures and vulnerabilities. Firstly, responsibilities directly dependent on the Site Operator are not discharged. This includes the failure to establish the CESC (Central Emergency Support Centre; the support group at British Energy headquarters) and shortcomings in alerting nearby residents to the incident (this process is partly automated, and so it is unclear if additional activity is required by the site operator after first declaring an emergency). Some failures at this stage are not problematic - the disabled site operator cannot distribute emergency planning leaflets, but these are distributed in advance of an accident, not during one.

Second-order effects from this failure are surprisingly limited. The Site Operator has no interactions or shared responsibilities with other actors, and does not produce resources that are needed by others. This is primarily due to the off-site nature of the plan and model - the site operator is responsible for a great deal of important activities during an emergency (safety of staff, on-site firefighting, reactor shutdown and containment etc.) which are purely internal to the power plant and hence do not feature in the plan or model.

None of these effects were explicitly reported from observations of the field exercise. CESC was not mentioned during any stage of the exercise, although it is possible that it was eventually established by different means (as communications with British Energy offices resumed later in the exercise). The actual activities of CESC during the emergency do not feature in the emergency plan (only its establishment), so the lack of mentions may reflect a lack of clarity over CESC's intended duties during the emergency. Likewise no explicit mentions were made of the emergency communication systems for local residents. This was not tested in practice

during the exercise, but testing of the communication and control systems for it would be useful. Reports from the exercises suggest that communication problems are not uncommon, and the lack of clarity over triggering the warnings could be a serious hindrance if these problems occurred in a real emergency.

These results correspond with earlier analysis of the plan-based model - there are gaps, ambiguities and irrelevant elements in the plan that made understanding and operating the plan difficult. Without access to the detailed plans of individual agencies it is not possible to determine if these problems are shortcomings of the plan itself (due to excessive abstraction, lack of clarity etc.) or if they are problems with the emergency response itself. In either case a process of review and revision would be beneficial.

6.8.2 Exercise Kilchattan

Exercise Kilchattan simulated an spillage of radioactive waste from storage tanks at Hunterston A, exacerbated by the failure of water drainage pumps. The response was hindered by storm damage blocking the main access route, and the need to handle a sponsored dog-walk occurring around the site perimeter. This was simulated by disabling responsibilities that required close access to the plant site - Establish Forward Control Point and Establish Traffic Control Points.

Disabling these responsibilities has almost no effect on the model, as they are not relied upon by any other entities. In theory this reduces the load on the police (who hold the most responsibilities within the plan) by eliminating two of their responsibilities. However, in practice this is more likely to increase their load as they abandon planned responsibilities and have to reallocate resources temporarily. Additionally they need to coordinate removal of the storm damage and re-establish access to the site. The dog-walking is also likely to place additional load on them, although is not modelled directly, as it is unclear who holds primary responsibility for this activity.

These overload issues did not arise in reports from the exercise; as with Exercise Indus, issues reported were more general problems with inter-agency communication and the structure of the exercise. However, neither exercise was particularly suited to investigating potential vulnerabilities in the off-site emergency response due to the choice of scenarios used. Both scenarios featured unexpected failures either directly on-site (pump failure, staff problems) or directly connected to the site (access roads, site communication links). These provide good opportunities to test on-site performance under degraded conditions, but do not provide a chance to test the robustness of the off-site response - in both scenarios the off-site agencies responded in a textbook manner without any deliberately induced failures. This may reflect a gap between the motivations of the site operators (who are primarily concerned with testing

their own procedures) and the external partners in these exercises; the report from Exercise Indus hints that a lack of detail in some parts of the exercise could be due to British Energy not caring about off-site elements [173].

6.9 Conclusion

By examining the Hunterston Emergency Plan and modelling it using formalised responsibility modelling we have been able to provide useful analysis while comparing our approach to the previous work by the InDeED consortium.

The analysis identified several shortcomings within the plan. These included vulnerable, critical chains of responsibilities as well as overloaded actors and unproduced resources. Many of these have been validated as valid issues by observations of emergency exercises or by their correction in more recent versions of the emergency plan. Additionally, those not directly validated by experience are still reasonable causes for concern; the lack of emerging problems due to them in field exercises reflects the operation and scope of the exercise rather than any fundamental irrelevance of the concerns.

The model produced broadly corresponded with the early work by InDeED, although InDeED modelled to a lower level of abstraction. The advantage of the refined responsibility modelling technique is clear - while InDeED's analysis was limited to separate models of individual areas, our analysis comprehensively covers the entire system and is guided by automatic analysis methods, providing valuable consistency.

In particular, management of communications and communication methods was identified as a key area where vulnerabilities could occur. A useful extension to the present model would be to explicitly model communication methods as resources, and use these resources to assess criticality. This is likely to be a useful technique in general when studying systems that rely on complex, multi-agency responses.

Toolkit-based simulations were also used to complement existing field exercises used to test the operation of the plan. These simulations generally revealed separate and contrasting vulnerabilities to those identified by observing field exercises, providing a low-resource method for identifying plan areas in need of refinement.

This chapter has set out a re-implementation of a previous InDeED modelling exercise. Our revised technique has demonstrated its suitability for modelling large and complex systems and generating useful automatic analysis from this model; the results show a strong similarity to previous work while improving considerably in the quality and consistency of the analysis.

Chapter 7

Case Study - Airborne Collision Avoidance System

7.1 Introduction

In this chapter we will model and analyse the use of the TCAS (Traffic Collision Avoidance System) aircraft collision avoidance system. This case study provides an opportunity to study a complex socio-technical system that has not previously been examined with responsibility modelling but that has been subject to substantial study and analysis in the field of aviation safety. This study allows the examination of the effectiveness of formalised responsibility modelling when applied to a new domain. In particular, it demonstrates that responsibility modelling can be used to capture the complex behaviour of socio-technical systems and that the methodology and notation cover the full range of interesting system elements.

The choice of domain is a complex socio-technical system, where a computerised alert and guidance system (TCAS) is used to provide additional protection against potential errors made by pilots and air traffic controllers. The combination of human and technical factors can lead to complex error states that would not occur if they each operated independently, and this may pose a threat when using TCAS. We will model and analyse the TCAS ecosystem using the publicly available standards documents and operating manuals and then validate via structured discussions with domain experts in order to detect inconsistencies, oversights and vulnerabilities within the operation or documentation of TCAS.

This study was conducted in co-operation with Sophrodyne, a socio-technical systems consultancy, which enabled the validation of the resulting responsibility model and analysis results with expert users of the TCAS system - a pilot and an air traffic controller.

TCAS was chosen as the choice of domain as it is well documented and studied system that exhibits substantial socio-technical features, but has not previously been examined using

responsibility modelling. Additionally, the expertise and access to domain experts of our industrial partner enables substantial validation of our models, which would be more difficult in other domains. This evaluation by domain experts provides an important understanding of how responsibility modelling is interpreted by users and how accurately it can capture complex system behaviour.

This case study demonstrates the benefits of formalised responsibility modelling as a modelling, analysis and elicitation technique. In particular, this study shows:

- the application of responsibility modelling to a new problem domain
- that responsibility modelling can be used as a technique to elicit additional domain information from system participants
- that formalised responsibility models can be generally understood by domain experts, without requiring extensive training
- that formalised responsibility modelling can accurately model TCAS, a complex socio-technical system, and its interactions
- that formalised responsibility modelling analysis results correspond broadly to documented issues, or issues encountered by system participants

The responsibility models produced during this study are available online in a GitHub repository: [159].

The remainder of this chapter is structured as follows. Section 7.2 provides the relevant background material for this study. This includes a discussion of TCAS's functionality and design, its integration within wider aviation operations and reports of incidents where TCAS was utilised. Section 7.3 describes the initial modelling of the TCAS system and environment, including the documents used and extensions to the standard responsibility modelling notation. Section 7.4 describes the analysis of this initial model by inspection and automatic analysis, alongside identifying relevant areas of interest not covered by the first version of the model. Section 7.5 describes the revised model developed based on the analysis results and study of mid-air collisions. Section 7.6 describes the methodology of the evaluation of the model and analysis results using semi-structured interviews with domain experts. Section 7.7 analyses the results of these interviews, considering the understandability of the model, the effectiveness of the analysis techniques and the accuracy and scope of the model. Section 7.8 examines other studies of TCAS and compares their structure and results with those of this study. Finally, Section 7.9 provides a final overview and conclusion.

7.2 Background

TCAS (Traffic Collision Avoidance System) is an automated warning system intended to reduce the number of mid-air collisions between passenger aircraft. The term TCAS is often used interchangeably with ACAS (Airborne Collision Avoidance System), which is an international standard that TCAS implements. TCAS uses transponder information from nearby aircraft to generate alerts and advisories; alerts provide the pilots with avoidance information, but responsibility for avoiding collisions still rests with the pilots. TCAS is estimated to have reduced the risk of mid-air collision by 80% [53] but complexities and confusion regarding the use of TCAS may have contributed to several fatal accidents.

These complexities arise for a number of reasons. While standard operating procedures insist that pilots follow TCAS alerts exactly, pilots retain the ultimate authority to act in other ways. TCAS alerts may conflict with previous instructions or procedures, and so pilots may choose to ignore them. In particular, the risk of collision may be increased (compared to not using TCAS) if the crew of one potentially intersecting plane follow the TCAS instructions and the other do not. In other cases, over-reliance on TCAS may hide fundamental problems in aviation or flight management. While TCAS has certain technical limitations, most potential vulnerabilities arise from the interaction between TCAS and the pilots, and between the pilots and other aviation actors. As a result, this problem naturally lends itself to socio-technical analysis.

Improvements to aviation safety can be made by increasing the performance and reliability of aircraft (such as significantly reduced engine failure rates), introducing safer and more reliable procedures (such as the increased emphasis on crew resource management) and providing automation where possible (such as improved autopilots, or the use of flight directors).

Some types of incidents are not easily resolved by automated systems, but can be (potentially) detected by such systems, with this detection used to inform the crew. For example, the widespread use of GPWS (Ground Proximity Warning System) has almost eliminated occurrences of controlled flights into terrain (where an airworthy, controllable aircraft is unintentionally flown into the ground) on suitably equipped aircraft.

Mid-air collisions represent a relatively rare but highly dangerous risk to aviation. While less common than accidents during take-off and landing or accidents caused by mechanical failure, mid-air collisions are almost always fatal to all those aboard. In controlled airspace (representing the majority of busy commercial aviation routes), collision avoidance is maintained by instruction from air traffic control, who instruct aircraft to fly on specific routes at specific altitudes and hence maintain *separation* - a safe distance between nearby aircraft.

However, this does not always prevent accidents. Air traffic control is normally based on instrument flying - pilots controlling the aircraft not by looking at the surrounding environment,

but navigating by the use of instruments such as altimeters (for height) and radio beacons (for direction). If this equipment fails, pilots may be unable to accurately determine their location, potentially bringing them into unsafe separation with other aircraft. Alternatively, pilots may be operating under visual flight rules, where pilots can navigate by observing the ground below and in most cases without control by ATC. In these cases, pilots are responsible for maintaining safe separation themselves, and collisions can occur if pilots do not accurately observe other aircraft. In particular, mixed airspace (where some parts are controlled, and others are flown visually) can lead to confusion and potential risk. ATC maintained separation is also vulnerable to small mistakes that can lead directly to collisions - such as clearing an aircraft to the wrong altitude (placing it at the same height as another) or breakdowns in communication with pilots (such as misunderstanding instructions, or reporting incorrect information). Alternatively, pilots may simply make mistakes, by for example drifting out of their assigned altitude, or inaccurately estimating the distance or speed of a nearby aircraft.

TCAS is intended to reduce these risks by adding an additional layer of protection. All large aircraft (and many smaller aircraft) are equipped with transponders, which broadcast information about their identity and altitude. TCAS works by interrogating the transponders of nearby aircraft, and using this to construct a model of the locations and bearings of nearby aircraft. By tracking aircraft over time, it is possible to identify aircraft that are on course to pass dangerously close to the TCAS-operating aircraft. If another aircraft enters the protection region around the TCAS-equipped aircraft TCAS will firstly raise a TA (Traffic Advisory), a non-prescriptive warning alerting the pilots to a potential risk. If the intruding aircraft flies further into the protection region (the predicted time to collision drops) then TCAS will issue a RA (Resolution Advisory) - a specific instruction to the pilots to manoeuvre the aircraft to avoid collision. TCAS RAs operate only in vertical space - an RA is either a command to climb or descend. In response to the pilots' actions, further RAs may instruct a faster rate of climb or descent, or instruct the pilots to reduce their rate of altitude change or level off as the risk reduces. While the pilots must manually make the manoeuvres required by TCAS the rest of the process is completely automated, and no ground-based radar or air traffic instruction is required.

TCAS operates as an additional layer of protection, and is not intended to replace separation maintained by ATC in controlled airspace. It relies on all nearby aircraft operating fully-functional transponders in order to provide the full range of protection - light aircraft flying without transponders are invisible to TCAS. As an advisory system, TCAS is ultimately reliant on pilots taking suitable action when an advisory is generated. Taking incorrect action can still lead to a collision.

TCAS originated from the desire of the Federal Aviation Authority (FAA, the US aviation regulator) to reduce the occurrence of mid-air collisions in the US. The basic principles and technology behind TCAS (transponder interrogation and threat tracking) were developed

as the Active Beacon Collision Avoidance System (BCAS) [194], which was eventually expanded and re-branded as TCAS. In response to further mid-air collisions the US Congress mandated the deployment of TCAS on medium and large aircraft (>30 seats) from 1993, which was later extended to smaller aircraft. At the same time, the International Civil Aviation Organisation (ICAO) was also supporting research and evaluation of collision detection and avoidance systems. The two separate tracks of development were integrated in 1997, with the launch of the Airborne Collision Avoidance System (ACAS) standard by the ICAO, and the launch of TCAS Version 7, which implemented the ICAO's ACAS standard. In 2003 an similar obligation to the earlier US mandate was introduced by the ICAO, requiring all aircraft with greater than 30 seats and with a recently issued type certificate[1] to be equipped with ACAS when travelling internationally. Aircraft flying purely domestic routes are exempt, but countries or supranational bodies may enforce restrictions similar to those for international travel, such as the US and the EU.

The deployment and operation of TCAS is governed by a range of documents, procedures and regulations. The ICAO provides an ACAS manual, as well as operational guides for aircraft operators and air traffic management providers and training materials for pilots and controllers. ICAO standards and recommendations, including safety standards and required equipment are detailed in the various ICAO Annexes. Additional regulatory materials are be provided by national or pan-national authorities (e.g. Eurocontrol, FAA). The three TCAS manufacturers (Honeywell, Rockwell Collins & ACSS) provide support and documentation for their equipment, and aircraft operators and air traffic managers provide standard operating procedures for their pilots and controllers. The public availability of these documents various substantially. The ICAO and national regulators publish their documents and notices in an open manner, while access to manufacturer and operator materials is restricted by commercial pressures.

7.3 Modelling

Modelling of the TCAS system followed the iterative modelling and evaluation strategy described in Section 3.7.2. Firstly, ICAO manuals were used to construct an initial model of the TCAS system and related actors and activity. This initial model was then analysed using responsibility modelling's automatic analysis techniques. Accident reports on major mid-air collisions were then used alongside these analysis results to refine the model. This revised model was then evaluated with domain experts (a pilot and an air traffic controller) who examined this model and the analysis results produced by the RESME toolkit. Throughout

[1] In effect, this requires all new or upgraded aircraft types to fit ACAS, while allowing older models to continue to operate.

Chapter	*Title*	*Content*
1	Introduction	Document overview and intention of ACAS
2	Implementation of ACAS	Obligations to install and use
3	Functions and capabilities	ACAS logic, displays and performance requirements
4	ACAS performance, safety and airspace configuration	Crowded airspace and pilot reaction times
5	Operational use and pilot training guidelines	Pilot training and operational procedures
6	Controller training guidelines	ATC processes, phraseology and training
7	Special uses of ACAS	Interactions with military aircraft
8	Safety and electromagnetic environmental assessments	Risk assessments and radio interference
9	ACAS performance monitoring	ACAS monitoring processes
10	ACAS-related transponder performance monitoring	Transponder problems
11	Certification and operational approvals	Approved systems and organisational processes

Figure 7.1: Contents of the ICAO ACAS Manual [84]

the modelling process the scope of the study was focused on the events just before, during and immediately after a TCAS alert, and entities outside that time period were only included if they explicitly affected events within that period. Additionally, the models were frequently discussed with the industrial partner, Sophrodyne, who provided feedback at regular intervals.

Modelling began by considering key elements from the ICAO manual describing the overall function and operation of ACAS [84], as TCAS is the only system that currently implements the ACAS standard. Specifically, this document refers to ACAS II (equivalent to TCAS 7) - the first version of ACAS to incorporate both warning Traffic Advisories (TAs) and instructive Resolution Advisories (RAs). This particular manual is intended to 'provide guidance on technical and operational issues' for both operational and training purposes. It is aimed at users of ACAS, and is not intended to be a full technical specification for developing ACAS-compliant systems.

This document is extensive and covers both the technical functionality and the operational usage of TCAS. Figure 7.1 shows the structure of the ACAS Manual, indicating the main content of each chapter. The initial chapters provide primarily technical detail on the ACAS logic and the integrated components, such as interference-limiting techniques for reading transponder readings. Much of this is not relevant to a socio-technical perspective, but abstractions of the technical details in Chapter 3 are used to define the key activities performed by the TCAS system itself, such as threat detection and the generation of RAs and TAs. The technical details of the TCAS system are deliberately abstracted, as the focus of the study is to evaluate the efficacy of responsibility modelling for the analysis of the wider socio-technical aspects of the wider system. As a result we choose to treat the TCAS hardware as one unified actor, rather than as separate sub-components.

Chapter 5 describes the operational usage of TCAS, and is the source of most responsibilities relating to the how the pilots react to TCAS alerts (when they respond to alerts is the subject of the short Chapter 4, which provides performance and timing information). This includes details on how to respond to changes in RA indications, and how to react when RAs are in

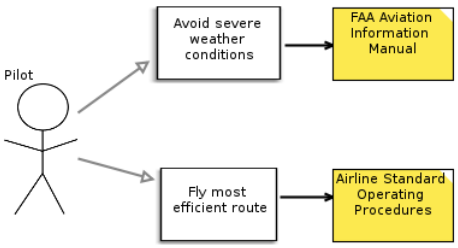


Figure 7.2: Responsibilities showing attribution to source

conflict with other instructions such as ATC instruction or GPWS (Ground Proximity Warning System) alerts or when they conflict with general operating rules.

Chapter 6 is focused on the operation of TCAS from the perspective of air traffic controllers, and how pilots responding to RAs should communicate with ATC. In principle this should represent exactly the same interactions as described in Chapter 5, merely discussed from the opposite perspective. However, certain ambiguities exist. For example, Chapter 5 states that pilots should perform the evasive actions indicated by the RA, and then inform ATC once completing the RA manoeuvres:

> "5.2.1.17 - If an RA requires a deviation from an assigned altitude, communicate with ATC immediately after responding to the RA."

In contrast, Chapter 6 states that pilot should inform ATC 'immediately' if their instructions are in conflict with the RA, presumably before completing the RA avoidance manoeuvres:

> "Table 6-1 - immediately inform ATC when they are unable to comply with a clearance or instruction that conflicts with an RA."

This small difference in wording opens up a period of time where the air traffic controller and the pilots have a different view of the world - the pilots are busy responding to the RA, while the air traffic controller believes all is well, as they would expect to be told immediately in case of trouble.

During the development of this model it became clear that attributing the sources of entities would be important, as even different chapters of the same document described differing and at times contradictory processes. Responsibility modelling does not have a notation for annotating or identifying sources is this way - previous versions have featured information resources, but these represent data sources *within* the model rather than those used to create the model.

To support traceability an extension to the responsibility modelling notation was introduced that does not affect any of the underlying semantics. Details on the origin of entities can now be modelled with 'Source' entities, which represent individual source materials such

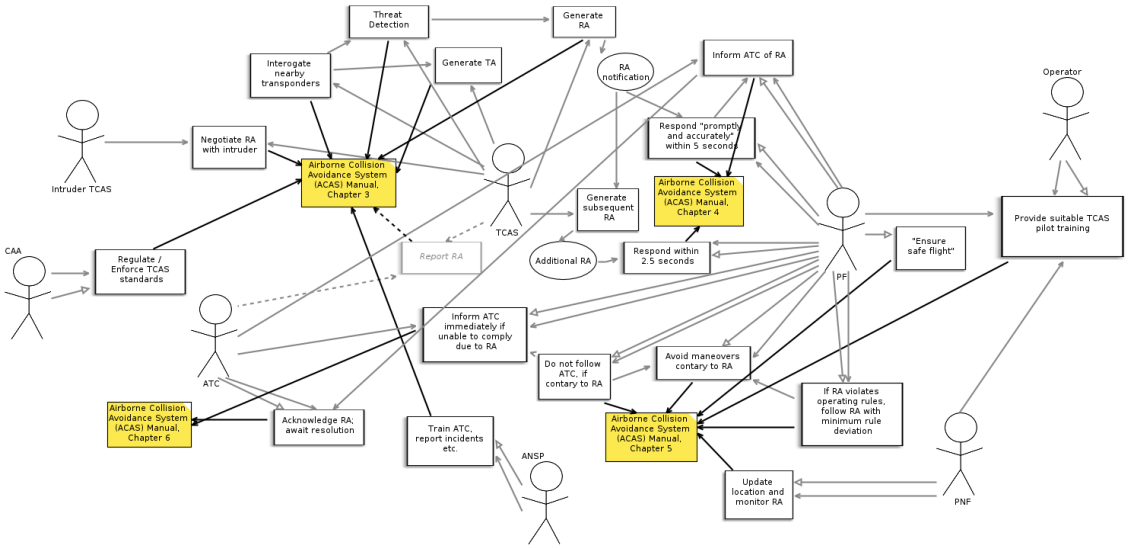


Figure 7.3: Initial responsibility model of TCAS

as documents (or parts of documents), interviews or field studies. Sources can be linked to responsibilities via ‘Attribution’ relations which indicate that a responsibility was defined based on details in the corresponding source materials. Only responsibilities can be attributed in this manner as in our experience different source materials differ more in the responsibilities they define than the resources or actors they list; attributing each actor or resource to multiple sources add further modelling complexity without a corresponding analytical advantage. This type of annotation can provide valuable information, enhancing traceability of model artifacts, providing a mechanism for linking responsibility models with other domain representations such as evidence artifacts in safety cases and enabling clearer indication of contrasting or conflicting responsibilities within a wider system.

Figure 7.2 shows an example containing the use of attribution. The pilot has a duty to avoid severe weather conditions (in the interest of safety) placed upon them by the regulations of the Federal Aviation Authority - the US aviation regulator. They also have a duty to fly the most efficient route between origin and destination (in the interests of speed and cost saving), as defined in the operating practices of their airline. These two responsibilities come from different sources and indicate a potential conflict of interests - if the most efficient route involves flying through patches of severe weather, what decision should the pilot make?

Drawing on the content of Chapters 3-6 produced an initial draft of the TCAS environment model, shown as Figure 7.3. This initial version can be accessed in the case study GitHub repository [159] as ‘TCAS.responsibilitymetamodel’ in the release ‘Initial Model’. This is a complex model that is not easily modularised, with many relations crossing between different sets of actors and responsibilities. The aircraft pilots and responsibilities performed by them are located on the right side of the model, with ATC activities in the bottom/left and operation of the TCAS itself in the top-centre. The TCAS unit generates alerts based on interrogating

the transponders of nearby aircraft, which are then displayed to the pilots, who must must respond to them correctly within strict time limits. The pilots also need to remain in contact with the air traffic controllers, and through the event still hold their basic duty of ensuring safe flight.

Responsibilities are also laid out based on the particular chapter of the ICAO manual they originate in, travelling clockwise from Chapter 3 in the top right. Alerts (in this version, only RAs) are represented as information resources that are generated by one responsibility and responded to (consumed by) another. Most interdependencies between systems and processes are represented by required responsibility relations, showing the direct reliance of one section on another, rather than creating arbitrary input and output resources.

Visual inspection of the model shows heavy load on the Pilot Flying (PF), who is required to interact with ATC and TCAS while following operating procedures and guidance issued by their operating and in compliance with ICAO standards. The TCAS unit generates appropriate alerts, communicating with other aircraft's TCAS units where necessary. ATC operatives only need to acknowledge pilots indication of RAs, and wait for the situation to resolve. Clear training and oversight roles are defined, including the CAAs (Civil Aviation authorities - the national aviation regulators), ANSPs (Air Navigation Service Providers - the flight traffic managers, including services such as ATC) and the aircraft operators. One part of the system is explicitly disabled, as indicated by the grey lines and fading - the automatic reporting of RAs by TCAS to air traffic control. This is defined in the ICAO manuals as a standard part of ACAS operation, but has never been deployed in practice and no set of operating procedures exists.

7.4 Initial Automated Analysis

Automated analysis of this initial version of the model does not reveal significant further information beyond that contained in the source documentation. There are no resources (alerts) that are not produced by a corresponding responsibility, and all resources are appropriately consumed and acted upon. No actors are defined but lack responsibilities and all responsibilities are assigned to actors, with one subtle exception - the overarching responsibility "Ensure safe flight" is held by the pilot flying, but the actors or resources required to discharge it are not defined. This reflects the disconnect between responsibility and action - the pilots are ultimately held accountable for the safety of the aircraft with minimal exceptions (massive mechanical failure, for example) but cannot ensure this themselves, as they implicitly depend on a wide range of other actors to maintain safety.

Load analysis shows that two actors may be overloaded - the TCAS unit and the Pilot Flying (PF). The overload warning on TCAS itself is somewhat misleading - this doesn't represent

the potential large volumes of data that can impair TCAS performance (a known concern in extremely busy airspace, and a justification for limiting TCAS deployment to larger, fixed-wing aircraft) but is instead triggered by the subdivision of TCAS activities into individual responsibilities (threat detection, transponder interrogation, RA negotiation etc.). These are all integral parts of basic TCAS operation, for which the system is developed. This level of load implies that TCAS is a complex system with many functions, rather than implying that it will be unable to function. This warning could be addressed by separating the TCAS actor into sub-components (communication, tracking, alerts) so that the actor definition is at the same level of abstraction as the responsibility definition; this is more elegant in theory, but TCAS is treated as a single unit by the source documentation, and failures of individual TCAS components are not widely reported.

The risk of overload on the pilot is much clearer - they are required to respond to various alerts and communicate with ATC, and are required to do so within very short timeframes. This is in addition to their normal duties of actually flying the plane (not expressed in the model), which is often a complex task; in particular, the situations that require TCAS response may often be those that are more complex in general, such as flying approaches in busy airspace. In practice, this is likely to lead to delayed responses to certain duties as the pilot attempts to prioritise the most urgent tasks, which opens up a number of vulnerabilities - for example, the mid-air collision at Uberlingen might have been averted by more timely communication between the flight deck and ATC; busy or distracted pilots have been a contributing factor to many accidents, and one of the major arguments for increased automation is to reduce the load on pilots [145].

Reliance analysis detects the high level of interdependency within the operational system. Civil aviation is a collaborative system, where each participant shares the duty of ensuring safety. Regulators enforce overall standards, but safe operation is ultimately based on participants choosing to operate and interact in as safe as manner as possible, even if this may be sub-optimal according to other metrics. Pilots rely on ATC providing them with safe routings, and ATC relies on pilots accurately and quickly following their commands, even though both can make reasonable claims to have the ultimate authority. On the edges of the system, some actors do not depend on any others - for example, aviation regulators (the CAAs) can complete their enforcement of standards without other actors in the system.

The organisational actors behind the individuals within the system - the air navigation providers and the aircraft operators - sit in a middle ground, being dependent on their employees (the ATC staff and the pilots) to participate in training and act as instructed, but rely on no third parties. Again, this is a slight abstraction of reality (for example, some or all training may be delivered by external training companies) but an appropriate one, as these details are unlikely to be of direct relevance (the content or quality of training may be relevant, but the nature of the provider less so).

In the core of the system there are high levels of mutual dependence. Both the PF and the ATC depend on each other in order to discharge their own responsibilities. The pilots and ATC also depend on the correct functioning of TCAS. The pilot, responds to generated TCAS alerts, whereas the ATC relies on TCAS indirectly when they temporarily suspend direct control while the pilot manoeuvres in response to an advisory. TCAS itself is not modelled as relying on any other actors - as a computer system, it cannot ultimately be held responsible for any actions and hence does not need other actors to avoid responsibility. This abstraction hides a significant part of the cause-finding / blame-apportioning process. The TCAS unit itself cannot be held accountable, but the manufacturers or maintainers of it could be.

Criticality analysis shows that the TCAS unit is the most critical actor and the most severe single point of failure (being ultimately required for 10 responsibilities, ahead of 9 for the PF). This is natural for a model of TCAS - without TCAS there can be no alerts, and so there can be no responses to advisories, RA-related communications etc. Similarly, the RA notification is the most important resource, triggering the response from the pilots (and later, ATC). The most critical responsibility is the 'Interrogate Nearby Transponders' functionality of TCAS, which is required in order to generate any of the advisories. The entire process is sequential (Scan airspace, generate alerts, respond to advisory, inform ATC) so the earliest inputs consequently have the greatest criticality factor.

This initial analysis shows that the model conforms to expectations, with vulnerabilities in the model representing well-known potential risks such as TCAS equipment failure and pilot overload. The appropriateness of the model's abstraction varies across different parts of the domain - at times it may abstract away important elements (such as the obligations on the manufacturers of safety-critical equipment), while other areas may feature unnecessary detail (modelling the TCAS analysis process as a series of separate responsibilities does not appear to have added any useful insight). Ultimately, the correct level of abstraction depends on the intended purpose of the case study, and for this study we have chosen to focus on the immediate short-term aspects of TCAS.

In order to refine the scope and abstraction of the model, an inspection was conducted of the 11 mid-air collisions that caused 100 or more fatalities using the Aviation Safety Network [60], for which a brief summary is given in Table 7.4. Given this high causality threshold, the majority of collisions in this sample occurred between large passenger aircraft. However, three of the eleven accidents occurred in collisions with military aircraft, and two with light private aircraft. These types of collisions may be more prevalent in the wider range of mid-air collisions, as accidents involving only one commercial aircraft are likely to have a lower number of deaths.

The two most common types of accident were collisions caused by failing to see another aircraft in VFR conditions, and collisions caused by ATC problems - either failing to follow

Flight Details	*Cause*	*Year*
TWA Flight 2 / United Flight 718	Collision under VFR	1956
United Flight 826 / TWA Flight 266	Deviation from ATC instruction	1960
MEA Flight 265 / Turkish Air Force	Collision under VFR	1963
ANA Flight 58 / Japan Air Self Defence Force	Military aircraft under instruction	1971
BA Flight 475 / Inex-Adria	Improper ATC operation	1976
PSA Flight 182 / Private Cessna	Collision under VFR	1978
Aeroflot Tu-134 x2	ATC error	1979
LAA Flight 1103 / Libyan Air Force	Disputed	1992
Iran Air Tours / Iran Air Force	Parallel takeoff/landing	1993
Saudia Flight 763 / KA Flight 1907	Deviation from ATC instruction	1996
GOL Flight 1097 / Private Embraer	Systematic ATC failure	2006

Figure 7.4: Features of mid-air collisions with fatalities > 100

issued instructions, or by following inaccurate or poorly communicated instructions. TCAS should be able to act as an additional safeguard in these circumstances, protecting against oversights by pilots or ATC. Indeed, the only accident on the list to occur after the international adoption of TCAS involved a faulty transponder on one of the aircraft, rendering it invisible to TCAS. However, this is more complex in situations involving military aircraft or private light aircraft.

Light aircraft are not subject to the same level of regulation as larger commercial aircraft, and may not be required to operate TCAS. In particular, large casualty accidents have occurred where commercial aircraft at low altitudes (usually on takeoff or approach to landing) collide with general aviation aircraft operating on visual flight rules. This is a subtly different scenario to the potential loss of separation between TCAS-equipped aircraft: in many cases general aviation aircraft will be fitted with transponders, allowing their detection by other TCAS-equipped aircraft but not receiving advisories themselves; in other cases they lack almost any electronics (especially very light aircraft such as microlights and gliders) and are effectively invisible to TCAS. This requires pilots to be actively aware of the surrounding airspace and not rely solely on TCAS to detect threats, which is not incorporated in the current version of the model.

Similar risks occur in interactions between TCAS-equipped aircraft and military airplanes. Military aircraft are fitted with transponders, but these may not provide altitude information, making accurate prediction of their trajectory more difficult. Additionally, TCAS is explicitly not designed to handle the extreme climb rates of military jets, which can lead to inaccurate advisories. Pilots attempting to respond to these advisories may be perceived by military pilots as flying erratically, causing further difficulty in ensuring safe separation. This situation can be further exacerbated in many countries by the separation of civil and military air traffic control - civil and military aircraft may be completely unaware of each other until visual

contact is reached, seriously limiting the reaction time and the ability of pilots to take evasive action.

7.5 Revised Model

Based on the results of initial analysis and the accident study a revised version of the model was developed, and is shown in Figure 7.5. Changes to the 'core' elements of the model (the generate-respond-inform approach to RAs) are minimal, with the main changes aiming to restructure some of the more peripheral areas to abstract away detail where considered unnecessary, and add in additional detail where analysis showed this to be useful. These include reducing the complexity of areas which caused automatic analysis to generate unnecessary warnings, and expanding the detail of areas that contributed to major mid-air collisions (such as VFR flight and ATC procedures). This revised version of the model can be accessed in the GitHub repository [159] as 'TCAS.responsibilitymetamodel' in the release 'Revised Model'.

The internal operations of TCAS were simplified by removing the 'Threat Detection' responsibility and integrating it in to the 'Interrogate Nearby Transponders' responsibility, which is now considered to hold all the internal logic of the system. These calculations are not particularly relevant to a socio-technical view of the domain, and do not represent a common point of failure.

The modelling of Traffic Advisories (TAs) was made more detailed, by adding both a resource to indicate a TA alert ('TA Notification') and a responsibility held by the pilot to react to this 'Identify Nearby Aircraft', which is an abstraction of the instructions given to pilots in Chapter 4 of the ACAS manual. This aims to provide a more consistent treatment of alerts, and eliminates the anomaly where TA generation was modelled as a responsibility, but TAs were not generated or reacted to.

The need for pilots to perform visual observations (especially when reacting to RAs, as they cannot then rely on ATC-induced separation) has been added to the model as the 'See and Avoid' responsibility, which is a duty on both pilots that is a subset of their general responsibility to ensure safe flight. This is derived from the same sub-section of the ICAO manual as 'Ensure Safe Flight' and can be discharged by the pilots alone; it contributes to ensuring safe flight, but that responsibility still cannot be discharged by just the pilots. The other main issue raised by the analysis of major accidents, military aircraft, cannot be addressed by revising the model as this is a fundamental weakness in the domain (fast moving, agile military jets can confuse both TCAS and pilots) and not an oversight in the construction of the model.

Additional details have been added to the workflow of air traffic controllers by introducing the STCA (Short Term Conflict Alert) [54] system and its corresponding alerts, as well as ATC

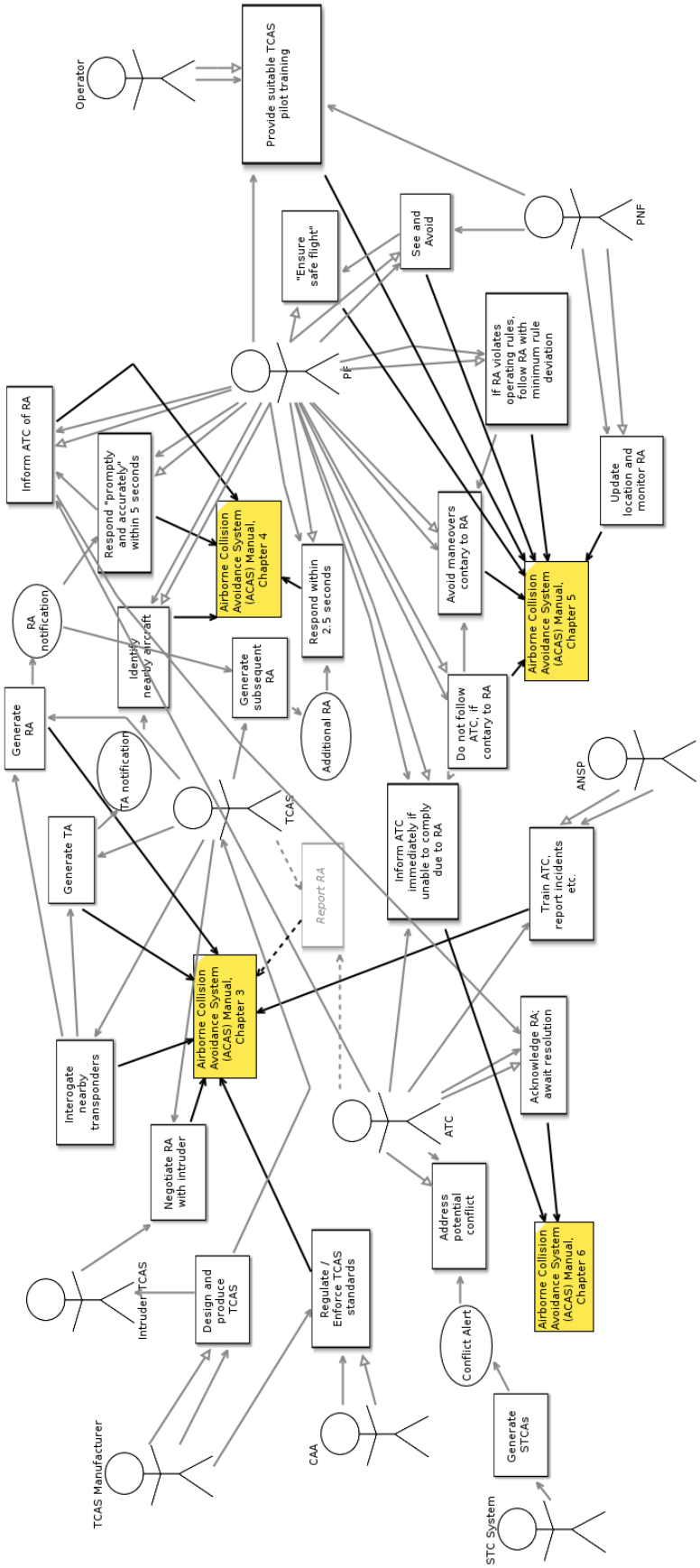


Figure 7.5: Revised responsibility model of TCAS

actions in response to them, in conjunction with pilots. STCAs are generated by computer systems that analyse aircraft positions and headings to detect possible loss of separation incidents, at which it point it generates an alert to the controller. In some ways STCA can be considered the ATC equivalent of TCAS - they both generate warnings ahead of loss of separation, but with some important differences. STCA alerts are purely advisory and do not indicate a potential resolution, requiring the controller to choose their own course of action. STCAs 'look ahead' from current positions up to around 2 minutes into the future, and have the potential to overlap with TCAS alerts responding to the same potential incident.

The most significant change is the introduction of the actor 'TCAS Manufacturer', which is responsible for designing and producing the TCAS units used in planes. This addresses the shortfall of accountability in the previous model - errors in the design or manufacture of TCAS equipment are serious issues where there is a need to apportion responsibility or blame, but it is meaningless to hold a technical component accountable for a fatal accident. The manufacturers are modelled as producing the relevant equipment (using the Produce Actor relation introduced in Chapter 6), and so a failure caused by TCAS itself can be traced back to its origins. Additionally, this allows a clearer expression of the the CAAs duty to enforce the TCAS standards by requiring the participation of the manufacturers in the discharge of that responsibility.

7.6 Interviews

In order to validate our model and methodology a series of interviews were conducted with domain experts to discuss the model and the results of our earlier analysis. These interviews had two main purposes. Firstly, to determine if the responsibility model of TCAS accurately represented the realities of TCAS operation and air traffic management, and if the concerns and vulnerabilities identified by the analysis corresponded to real-world problems. Secondly, discussing the models with domain experts would test the understandability of the model, and that of the core responsibility modelling concepts in general. This portion of the study was approved by the University of Glasgow College of Science & Engineering Ethics Committee under application number 300150172.

Two interviews were conducted with experienced experts representing the main actors in the systems - an air-traffic controller and a pilot. Firstly, Controller M was interviewed, an air-traffic controller with ten years experience as an area controller. Secondly, Captain A was interviewed, a pilot with fifteen years experience, including as a training and check captain.

The interviews were initially designed as semi-structured interviews [66] using the responsibility model and analysis results (presented using the RESME toolkit) as artifacts to discuss. Interviews began with a short presentation of the responsibility modelling notation and an

introduction to the toolkit using a worked example. Participants were then shown a series of sub-models representing different parts of the system (ATC, TCAS & Pilots) and asked to write short descriptions of them (to test the model's understandability). For each sub model, participants were asked to discuss the similarity between the model representation and their experience of these sub-systems in the real world; prompted points of discussion included correctness, scope, level of abstraction and relevancy. After examining each submodel participants then inspected the whole model, with prompted focus on interactions between sub-systems and the identification of missing elements. The interviews then concluded by discussing the model-wide analysis results with the participant - the results of each type of analysis were introduced and presented, and the accuracy and relevance of these warnings were discussed. Participants were also asked to identify any important vulnerabilities that they felt had not been detected.

Interviews were recorded using the Camtasia software, which records both the device screen and the voices of the participant and interviewer. These recordings were then transcribed by the interviewer, and full transcripts were deposited in the University of Glasgow's Enlighten service [160].

The case study was piloted using a representative of the industrial partner as the interviewee. The interviewee has a background in aircraft accident investigation and was involved in initial discussions of the project, but had not seen the final model and was unaware of the structure of the interview in advance. Several issues were identified during this pilot study. The introduction of the technique was limited by the use of a worked example in a domain that was unfamiliar (the Scrum model of software development) and that did not contain all types of relation. During the pilot the decision was taken for participants not to write descriptions of the sub-models, but instead to describe them verbally, which led to a more natural and faster-flowing interview. Some issues around terminology were also identified, such as the differences between responsibilities and goals or tasks and the need to emphasise the relationship between actors and roles. However, the overall structure of the interview appeared to deliver good opportunities to identify potential issues and the supporting software (the RESME toolkit and the Camtasia screen and voice recorder) worked without difficulties.

As a result the interview plan was restructured to eliminate written descriptions and make fuller use of voice and screen recording by asking the participant to verbally describe the model instead. A new worked example was created around the taking of orders and cooking of food in a restaurant, which demonstrated the full range of relations in standard responsibility modelling.

Between the first and second interviews a small change was made to the interview structure - additional prompts were added to emphasise the initial description of each model by the participant. In the first interview Controller M's first response to each model was often to ask

questions and resolve ambiguities, which then led naturally to discussion of individual points rather than getting a brief overall view of the model and its general understandability. This change helped to ensure that clearer descriptions were provided by Captain A in the second interview.

7.7 Interview Results

Firstly, this section discusses the participants' understanding of responsibility modelling, and highlights parts of the notation that appeared more difficult for them to understand and use correctly. Secondly, issues around the scope of the model are examined, and whether this captures the full range of TCAS-related interactions described by the participants. We then discuss the implications of each form of responsibility modelling analysis by describing our evaluation of the results with the domain experts. This is followed by an analysis of the particular problems caused by the intersection of roles and ranks in this particular environment and a discussion of emergent behaviour described by the participants but not documented in the official TCAS standards. Finally, the interviews are evaluated as means of model and analysis validation and we consider whether it is possible to evaluate both the understanding of a technique and the validity of a model in the same experiment.

Understanding

The difference between the two different actor-responsibility relations (Required Actor and holds) was the most problematic element of the participants' understanding of responsibility modelling. This manifested most obviously when one of these relations was present without the other; this often led to confusion over what that relationship indicated, such as the idea that it was possible to be accountable for a responsibility without having any involvement in discharging it. Both participants' understanding of the model itself was good, although Controller M took significantly more time to process each part of the model before describing it or asking clarifying questions.

Controller M appeared to have a slightly incorrect interpretation of the distinction between holding a responsibility and being required for a responsibility - they described *"the solid black arrow [Required Actor] is they are responsible for that, but the empty one [Holds] is that they're also accountable for that"*. This is a correct definition of Holds (indicating responsibility and accountability), but is not strictly correct for Required Actor - it is possible to be required for a responsibility without any concept of being responsible (in the sense of duty, liability etc.) for it.

Controller M also asked several questions to clarify their understanding when presented with each part of the model. These generally related to a lack of detail in the short description of each responsibility - for example, he asked for more detail on 'Update location and monitor RA', which described the Pilot Not Flying's duty to keep track of the plane's location, the positions of nearby aircraft, the status of the RA etc. Using relatively short names for each entity is important to maintain the 'first glance' understandability of the model and to minimise its size, but it may be beneficial to include more detailed descriptions, such as via a notes tab for each entity.

Captain A often used the phrase *"lack of direct control"* to refer to cases where an actor held a responsibility, but was not a required actor for it. Similarly he occasionally used 'direct control' to describe the Required Actor relation. These descriptions do not strictly match the meaning of the Required Actor and Holds relations, but they are a reasonable approximation. A Holds relation (without a corresponding Required Actor) does imply a lack of a direct control, but that is only part of the relation's meaning - most importantly, the holds relation indicates a sense of authority or accountability with regards to a responsibility. Similarly, describing a Required Actor relation as 'direct control' is only partially accurate. An actor required by a responsibility is directly involved in that responsibility - without them it cannot be discharged. However, being required for a responsibility does not imply that they have any control over that responsibility; responsibility modelling does not explicitly feature the concept of control, instead using functionality (required entities) and accountability (holds). It is possible for a required actor to have no control over a responsibility at all; they may be required only to complete a trivial task, or their mere presence or existence may contribute towards the discharge of the responsibility.

However, Captain A was also able to use the difference between these two relations to pick up on potential ambiguities within the model. He inquired as to how the Pilot Flying held but was not required for the responsibility 'Ensure Safe Flight' (in his words *"When you have Pilot Flying not having control of safe flight"*), where no required entities had been specified because of the abstract and overarching nature of this responsibility. This is an important subtlety, as the pilot is held responsible for safe flight, but the model does not indicate which agent is required to actually discharge the responsibility, as the actions required to do this are unclear. This could potentially lead to a unexpected responsibility failure, and it was reasonable to identify this as an anomaly.

Overall, he appeared to grasp the notation quickly, and had no difficulty providing quick and reasonably accurate descriptions of the models he was presented with. In particular, he noted *"It's quite easy to interpret and it's very familiar language to a pilot"* - the use of the correct technical terminology contributed to his understanding of the model. While Captain A made no comments that implied he was using his own experience to describe a situation rather than the model, the speed at which he described models and responded to questions suggests either

an extremely quick reading and comprehension speed or some use of pre-existing knowledge to interpret the responsibility models.

No observations were made about the RESME toolkit, and the graphical representation of the models did not appear to impede comprehension (with the exception of infrequent comments about overlapping relations, or entities laid in an non-intuitive way). The optional sourcing layer (showing which entities had been derived from which parts of the source materials) did not appear to add any extra information for the participants, and so was disabled to increase the clarity of the model.

Overall, the participants appeared to understand the model and the notation well, given the limited time they had to inspect and understand them. The notation itself was unproblematic with the exception of the difference between holds and Required Actor relations, which led to confusion over the different ways actors and responsibilities could be linked. The model was well understood, although short descriptions led to uncertainty over the full meaning of certain entities. Given that both participants were experienced experts in this particular system there is the possibility that other shortcomings in the model were hidden by their domain experience, although no obvious evidence of this (such as responses contrary to the details described in the model) was evident.

Scope

An implicit assumption made during the modelling process was that all activities described in the model were occurring in controlled airspace, where air traffic control monitors and controls the movements of aircraft. Most airspace in built-up areas and around busy commercial airports is controlled, but uncontrolled and procedural (where instructions are issued based on timing and routing, but controllers cannot track aircraft movements) airspace is widespread in rural areas and across large seas and oceans. Controller M noted that the general structure of the model would work in both controlled and uncontrolled airspace - *"I think this scenario would work anywhere"*, but with the role of air traffic control being reduced in procedurally-controlled airspace and uncontrolled airspace.

In principle it would be possible to model this type of operation in uncontrolled airspace by reducing the existing model to a subset - the interactions involving air traffic control are removed, leaving just TCAS and the pilots as the main operational actors. This approach requires no changes to the model and can be easily studied using 'What-if' analysis. However this does not accurately represent the reality of aviation operations in uncontrolled or procedural airspace - when air traffic control is reduced or removed pilots assume additional co-ordinating functions. Captain A described how they control their flights in rural Africa - *"you can't reach anybody on the radio, so again you're relying on your TCAS, you're speaking to other people, position reports"* and when operating out of uncontrolled airfields - *"we rely*

on TCAS to work out where we are with other traffic; we'll be talking with other traffic to organise our own air traffic control effectively". These additional tasks place extra load on the pilots and add extra complexity to the overall system, which may have indirect effects on their ability to respond to TCAS advisories. Additionally, these additional responsibilities may generate conflicts between TCAS instructions and other forms of guidance; for example a TCAS advisory may require pilots to deviate from a particular track in procedural control, which could potentially bring them into the path of another aircraft. As a result, it is clear that a cut-down version of the existing model would not accurately represent behaviour in uncontrolled airspace.

To model these different situations accurately would ultimately require multiple models - each model would share the same core entities covering standard pilot and TCAS activities, but would be augmented with the appropriate ATC or pilot co-ordination and planning responsibilities appropriate to that type of airspace. These augmentations will affect overload, reliance and other indicators of system safety and performance, so a new analysis is required for each variation of the model. Unfortunately, responsibility modelling does not provide a practical way for models to share a common core of entities, and instead requires manual duplication of the same entities across multiple distinct models. This makes maintaining multiple models impractical, as changes must be manually synchronised and each model risks unintentional deviation from the common core. As a result our responsibility model of TCAS can only be considered accurate for aircraft in controlled airspace, although many of the discussion points raised and vulnerabilities suggested may apply in other contexts as well.

Overload Analysis

Controller M noted that pilots were probably overloaded, but not to the extent that their performance was impaired - *"you're probably right that he's overloaded with responsibilities... I'm not sure it's to the extent that it would impair his ability to resolve the situation"*. In contrast, they felt that air traffic controllers were potentially overloaded to the extent that their performance was affected, especially under difficult conditions - *"you basically not only loose control but you loose awareness of what's actually happening"*.

Captain A generally noted that while pilots often had to consider many responsibilities their ability to react and respond promptly was not affected. For example, he noted that *"we have to respond within five seconds of an RA - that's actually quite a long time when you think about it"* and felt that maintaining the required levels of performance was not a problem. They also felt that controllers could become overloaded, especially in impaired conditions like bad weather, which led to delays and reduced responsiveness - *"we're listening to them and we're spending like five minutes just to check in and tell them you're speaking to them, and that's just the way it is"*.

Discussion of overload analysis highlighted the differences between human or organisational actors and technical actors. The RESME toolkit generates overload warnings for TCAS itself, as it holds a large number of responsibilities. However, interviews revealed that TCAS has no difficulties in tracking multiple aircraft and co-ordinating and generating multiple alerts simultaneously - in that scenario overload is more likely to affect the pilots, as they attempt to comprehend multiple warnings (Captain A: *"I've never doubted that TCAS can cope. ... the transponder will often be shouting at you - Traffic!, Traffic! and so you try and look for it and can't see it..."*).

This reflects a general difference between technical and non-technical actors - technical systems are usually designed to meet a certain performance level (in terms of throughput, simultaneous actions, processing capability etc.) which simplifies overload analysis to comparing expected demand to the technical system's specifications and performance; it is perfectly practical for such an actor to discharge a large number of responsibilities or perform multiple activities simultaneously if it has been designed for this purpose. In contrast, there are fundamental limitations on human performance in multitasking and working memory, while organisations are often limited by logistical or structural inertia. As a result, it may appear logical to differentiate between different types of actor when generating overload warnings.

However, the distinction between these types of actors is not explicit. The name of an actor may indicate its nature (e.g. 'Scheduling System'), or it can be inferred from the responsibilities held or that it is required for (responsibilities expressing accountability, authority or problem-solving are indications of a non-technical actor). The inherent flexibility of roles means that the implementation of an actor role by a specific entity may cross this distinction - for example technical systems may replace individuals within an organisation due to computerisation, or the failure of a specific computer system may lead to its duties being taken on by human staff instead. This has no effect on the responsibility structure as all relations remain the same and no explicit actor typing exists, but may affect the implications of any potential overload warning. Hence, attempting to determine the type of actor at the overload warning is impractical, but the status of the actor should be taken into consideration when interpreting such overload warnings.

The interviews also revealed unintuitive effects of load, where adding additional tasks actually reduced (temporarily) the effort and load required. Captain A noted that the generation of a TCAS TA placed additional load on the pilots, as they had to try and identify (and potentially manoeuvre to avoid) an intruder aircraft while still maintaining all their existing responsibilities (such as flying within the constraints set by ATC). However, responding to a TCAS RA was considered very simple, as almost all other tasks and duties could be suspended due to the importance of the RA; the only outstanding responsibility was to fly in the aircraft in a safe way (avoiding unnecessary stress), and other responsibilities such as fuel

management or navigation could be temporarily ignored.

This challenges the load and overload concepts used within responsibility modelling; adding an additional high load responsibility (such as responding to an RA) to an actor with other responsibilities should significantly increase the risk of that actor, as responsibility modelling uses an additive model of load. This in contrast to the described behaviour for TCAS RAs, where responding to a high load responsibility effectively disables other responsibilities for the duration of the response. However, this does not directly contradict the structure of load in responsibility modelling, although it can be counter-intuitive. Load (and overload) in responsibility modelling does not directly represent the instantaneous effort of performing a particular task, as Captain A expressed the effort of responding to an RA. Instead, load in responsibility modelling represents the organisational effort of monitoring, preparing for and evaluating responsibilities. This often correlates to the amount of effort required to actively perform a responsibility, but this correlation does not always hold, particularly in cases of high instant effort and reactive responsibilities like responding to an alert.

Overall, validation of overload analysis with domain experts provided a mixed result. Overload warnings for the main non-technical actors (Pilot Flying & Air Traffic Controller) identified actors that were definitely at risk of overload, but which differed in the experts' opinion of whether this risk would actually lead to impaired performance. This result fits within the framework of automated vulnerability analysis, which is designed to detect potential vulnerabilities and does not claim to identify guaranteed problems. However, the overload warning for TCAS was less useful, as TCAS is designed to handle large numbers of potential events and was not at risk of impaired performance due to load. Responsibility modelling's treatment of these technical actors provides flexibility, but may cause many technical actors to generate false positive overload warnings. Additionally, potentially counter-intuitive behaviour was discovered where domain experts observed that the net effect of adding complex responsibilities on load was minimal due to prioritisation of tasks. Overload analysis detects such actors as overloaded, but effective prioritisation minimises the negative impacts of this condition.

Criticality Analysis

Criticality analysis identified the TCAS hardware and the responsibilities and resources involved in RA generation as the most critical entities. Given the model focuses on the response to RA incidents this is an unsurprising result, but it was used to generate discussion around the possible failure modes of these critical components, and what effects that would have on other actors in the system.

Controller M emphasised that TCAS is a bilateral system since it relies on obtaining accurate information from other aircraft to generate alerts. The model shows the case where both

aircraft have a TCAS system installed, but it is also possible to successfully generate warnings where only one aircraft has TCAS, but the other has a Mode C transponder (a transponder providing altitude information). If a transponder is inoperable or provides inaccurate information this eliminates or reduces the safety benefits of TCAS, which already operates to tight margins of separation. Alternatively, a transponder without altitude information can lead to Traffic Advisories that warn of a nearby aircraft without identifying its location, which can disorientate the pilot and place extra load on the air traffic controller as they attempt to identify the nearby aircraft. Problems can also arise due to incorrect altitude values from transponders, which air traffic controllers may identify if the pilot's reported altitude does not match the transponder-derived altitude recorded by the ATC system.

Captain A discussed the implications of an absence of TCAS. TCAS-fitted aircraft can operate without it being functional for a limited period of time, subject to certain restrictions on airspace and the need to inform air traffic control. He noted the use of TCAS for a wide range of functions, including those outside of the core TA / RA functionality, such as using the traffic information screen to check for approaching aircraft before taking off. Without it, he felt that he was missing important information - *"It's a huge gap in your awareness"*. He also noted a particular unusual situation where TCAS could generate spurious alerts. Some ships are fitted with transponders, and these transponders can be interrogated by TCAS. Naval transponders do not transmit altitude, so TCAS is unable to generate advisories but does display this information on the traffic information screen. This alert can be difficult to distinguish from an aircraft flying nearby without altitude information, leading to confusion for the pilots - *"That's the only time that I've really encountered any kind of TCAS not being helpful."*.

In this case study the effectiveness of criticality analysis is fairly limited - the most critical entities are already apparent from the basic structure and scope of the system. This is a consequence of the model being built around one particular chain of responsibilities: the generation of and response to TCAS alerts. The potential strength of criticality analysis is its ability to identify the key points of more distributed systems or test the effects of system changes, but its utility in a centralised, unchanging system is minimal.

Reliance Analysis

The temporal scope of the model led to differences in the interpretation of the reliance between actors. Controller M raised the issue of the Pilot Flying relying on ATC *"in terms of an RA, I would sort of question that they are relying on the air traffic controller during an RA"* as they felt that the responsibility to handle an RA lies with the pilots alone. During the response to an RA the pilots are removed from ATC control until they successfully complete their response to the RA. However as soon as the response to the RA ends the pilot must inform ATC that

they are now back under their control, and this communication with ATC is effectively the end of the RA response process. As a result, whether or not the pilot relies on the air traffic controller depends on which timescale is being considered.

In contrast, Captain A accepted the full range of actors that the Pilot Flying relied on (ATC, PNF, TCAS, TCAS Manufacturer) and suggested additional actors not currently featured in the model, such as aircraft maintenance and airline trainers who are responsible for ensuring the TCAS and the PNF respectively are capable of operating to the correct standard. Additionally, they noted that they rely on nearby aircraft also having TCAS, which is particularly important outside of controlled airspace, as pilots must plan their own routes and manoeuvres with minimal external guidance. This also requires direct communication and reliance on other pilots in order to achieve mutual co-ordination; this is normally intermediated by ATC in controlled airspace.

Indirect reliance identified by responsibility modelling also demonstrated complex interactions in practice. Air traffic controllers are identified as relying on TCAS, which they never interact with directly. However, Controller M described a scenario where a pilot under ATC instruction also attempted to follow a TCAS Traffic Advisory; in doing so they flew their plane above the safe altitude level that they had been cleared to by ATC. While TCAS Resolution Advisories are explicitly intended to override ATC instructions TAs are for information only, yet if pilots follow them without considering their compatibility with existing ATC instructions a loss of separation can occur.

Throughout the model the high level of interaction and dependency between actors was evident. Captain A agreed that the response to TCAS (and aviation as a whole) demonstrated a high level of interdependency - the Pilot Flying and air traffic controllers both rely on almost every other actor modelled in the system. A number of reliance listings were considered self-explanatory and did not receive any substantial discussion by the interviewees , such as an Air Navigation Service Provider (ANSP) relying on air traffic controllers (their staff). However, all of these reliance links were considered reasonable, with no objections raised against them.

Overall, reliance analysis produced results that were considered valid and occasionally identified points of interest that were not obvious, but most reliance results were self-explanatory. In this particular case study that is not unexpected - the tight interdependency between pilots, their aircraft and air traffic control is already well understood, and responsibility modelling highlights this. While providing limited new insight none of the reliance relationships generated were identified as invalid, which provides confidence in the structure of the model and the reliance algorithm, and could produce more interesting results in a domain where interactions and dependency have not already been widely studied.

Roles and Ranks

Discussion of the differences in roles between the Pilot Flying (PF) and the Pilot Not Flying (PNF) raised the issue of dynamic role swapping within a system. Responsibility modelling actors represent roles, and so responsibility models (at least at the modelling stage) are not concerned with the assignment of individuals to actor roles. However, our previous discussions of roles in modelling and role analysis have made the implicit assumption that role allocation is fixed - roles may be played by multiple individuals or organisations, multiple roles may be held by individuals etc., but these allocations are stable and do not change during the operation of the system, which simplifies modelling and analysis.

However, our interviews highlighted that swapping of roles between the PF and PNF is a standard feature of commercial aviation - the PNF will step in if they feel that the PF is overwhelmed or is making incorrect decisions. Currently, the TCAS responsibility model features no representation of this swap - either as a specific responsibility, or as some form of actor allocation, and is unclear which approach is the best fit to address this type of change. Adding an explicit responsibility or responsibilities (i.e. 'Monitor PF' and 'Switch roles if required') provides a clear description of the potential behaviour, but potentially breaks abstraction by including role allocation information directly in the model itself. Alternatively, handling this role exchange in analysis only makes an important aspect of the system invisible, reducing the fidelity of the model in the eyes of stakeholders.

Additionally, the respective roles of pilots are further complicated by differences in rank. The captain of an aircraft (also known as the Pilot in Command) holds overall responsibility for the operation and safety of the aircraft [83]. Captains also have a supervisory relationship with more junior pilots, and are trained to advise or intervene if they make mistakes (Captain A: *"when ... the first officer is flying the aircraft I'll try and let him or her make all the actual decisions - however if they make a wrong decision it's my responsibility to correct them."*) This places additional responsibilities on the captain as they are required to observe the other pilot(s) even while they are themselves Pilot Flying, and are held responsible for the entire flight crew. The difference in ranks also influences the social dynamics of the PNF stepping into override the PF; several serious incidents have arisen when more junior pilots have been reluctant to challenge more experienced and more senior captains, which is an important focus of Crew Resource Management [33]. However, the ICAO ACAS manual makes no references to 'Pilot in Command' and just one to 'Captain' (in describing frequency of simulator training), implying that the issue of seniority (and any potential problems arising) was not considered of concern by the ICAO.

Responsibility modelling does not have an explicit means for modelling these differences between roles and ranks. Pilot in Command can be modelled as a distinct role, as it features distinct responsibilities that are attributed to it specifically. However, there are no specific

relations and responsibilities that First or Second Officers would hold or be required for, with the possible exception of supervision by the Pilot in Command. These roles could be used to construct a small sub-model showing the supervisory and oversight relations between senior and junior pilots, but this would be separate from the operational model of TCAS as described in the documentation. However, the rank dynamics expressed in that sub-model may affect the successful discharge of the main TCAS model, such as when the authority of the PF is undermined by them being a more junior officer. This influence of the sub-model cannot be explicitly modelled, and could lead to potential vulnerabilities going undetected. This occurs because of the lack of a formal way to relate the rank-based classification (Captain, First Officer) and the activity-based classification (Pilot Flying, Pilot Not Flying) of actors.

Alternatively, role analysis could be expanded to consider not just the interactions when actors are assumed by the same entity but to also consider how actors that are working collaboratively (such as being involved in relations with the same responsibilities or resources) relate to each other. This would enable potential vulnerabilities caused by incompatibilities such as uneven power balances to be analysed.

Emergent Behaviour

Our model of TCAS focuses on the use of TCAS to generate TAs and RAs - advisories that warn and advise pilots of potential collisions. This the core functionality of TCAS - as the ICAO ACAS Manual describes it: 'The objective of ACAS is to provide advice to pilots for the purpose of avoiding potential collisions. This is achieved through resolution advisories (RAs) ... and through traffic advisories (TAs)' However, Captain A described a number of situations where TCAS functionality (most commonly, the traffic information display) is used for flight planning, spatial awareness or co-ordination - none of which are explicit functions of TCAS. This demonstrates the importance of supplementing system documentation with descriptions of the system from stakeholders.

In busy airspace he noted that they used TCAS traffic information screen to locate nearby aircraft and potentially modify their route - *"we'll change the way we are ascending or descending to avoid the potential of becoming a TA or an RA [with another aircraft]"*. Similarly, they used the same information to manage their approach to airport holding - *"if we're going through a holding pattern and see lots of aircraft below us we know there is a lot of holding, so we'll slow down"*. TCAS was used in the same way when preparing to take off - *"we'll use it when we line up on the runway - we'll put the TCAS on and make sure there's nothing imminently on approach"*. Finally, he reported on the use of TCAS traffic information when in uncontrolled airspace as an alternative to air traffic control - *"we go into uncontrolled airspace, uncontrolled airfields where we rely on TCAS to work out where we are with other traffic"*.

This usage is given only a brief mention within the ACAS manual. The ACAS manual notes 'The TA display presents the flight crew with a plan view of nearby traffic. The information thus conveyed is intended to assist the flight crew in sighting nearby traffic.' and also that 'Continuous display of proximate traffic is not a requirement for ACAS.'. This discussion of displays consists of just three paragraphs within a 203-page document, but represents an important use of TCAS in practice.

Deviations between theoretical and real-world usage of systems are a common characteristic of socio-technical systems [11]. In this case TCAS is used for a much wider range of applications than is described in the standards documents, and hence also a wider range of applications than specified in the responsibility model. This difference has potentially problematic implications for the safe operation of the system if this unspecified behaviour is not supported to the same level as the specified functionality. For example, the TCAS traffic information display is only a minor component of responding to TCAS advisories, and so an inoperable or substandard display might not be considered a serious problem that requires a quick fix. However, the traffic information display is a key component of much of the unspecified behaviour performed by pilots, so its absence would have strong detrimental effects on the pilots' awareness, placing them under additional stress and load. Unless managed or recorded in a structured way these gaps in the specification of the system will result in a lack of support for important but undocumented functionality.

Suggested Model Improvements

These changes are limited to those within the scope of TCAS alert-related interactions within controlled airspace. Issues around the application of this model outside of that scope have been discussed in a previous section.

Both interviews suggested that our strict allocation of responsibilities between the Pilot Flying and the Pilot Not Flying was incorrect. Controller M noted that it could be the PNF that informed the air traffic controller of an RA; he did not see it as specific to either role. Captain A noted several responsibilities that the PNF was involved with - ensuring safe flight and ensuring the RA is correctly followed. The PNF could override the PF if they were not following the procedure. He noted that in practice these roles involved more flexibility than the model expressed, but that there was also an overlap between responsibilities being shared between pilots and pilots swapping the PF / PNF roles (as previously discussed in Section 7.7). It is clear that our strict interpretation of the ICAO regulations does not match actual practice, and that several responsibilities should be shared between both pilots. For example, 'Inform ATC of RA' might be discharged by either of the pilots, and both pilots are obliged to 'Ensure safe flight' (although the ultimate legal responsibility rests with the pilot in command). This is partially in conflict with some of the details described in the ICAO manuals, so additional

reputable information sources (such as airline training documents) would provide evidence that these changes are valid.

Captain A also mentioned a potential danger of strictly following a TCAS instruction if the aircraft is at the edge of its performance envelope - such as when it is flying at close to maximum altitude. TCAS logic prohibits descend instructions at low altitude, but does not prohibit climb instructions to the aircraft ceiling. Responding to such a climb command could put the aircraft above its certified maximum altitude, potentially causing it to stall and enter the dangerous 'coffin corner' [65], which causes stalls that are difficult to recover from. This potentially creates a safety obligation to avoid following RAs accurately if doing so would leave the aircraft uncontrollable - this could be modelled by rewriting the 'Respond promptly and accurately within 5 seconds' responsibility to 'Respond promptly and accurately... if safe to do so', or by introducing an additional responsibility for the Pilot Flying - 'Keep aircraft within performance limits'. However, the pilot may ultimately be forced to ignore one of these responsibilities to prioritise the other, such as when stalling the aircraft is the only option to avoid a collision. As discussed in the previous section, Captain A mentioned the frequent use of the TCAS information display for non-collision avoidance purposes, such as managing holding patterns or observing other aircraft in non-controlled airspace. This behaviour is an important yet sparsely documented application of TCAS, but does not fit our specific scope for this model, as it is not directly related to the the generation or response to TCAS TAs and RAs.

Controller M discussed the potential knock-on effects of TCAS, giving the example of a holding stack where large numbers of aircraft follow the same paths and are separated vertically. In this scenario, one aircraft deviating from its assigned height could trigger an RA on the aircraft above or below, which would then react and trigger an RA in the next aircraft in the stack, and so on. This highlights how the actions of one aircraft can have subsequent effects on other aircraft, and that a response that is optimal for one aircraft may adversely affect many others: Controller M mentioned that he would advise pilots of other aircraft in the stack in order to try and limit the consequential effects. In contrast to this, the responsibility model of TCAS considers the interactions between aircraft only in terms of whether or not a potential intruder has a functional TCAS system; there is no discussion of potential effects on other aircraft, and no discussion of the fact that a loss of separation may require responses from multiple aircraft.

The most comprehensive approach to incorporating this into the model would be to significantly expand the modelling of the intruder aircraft, such as by explicitly modelling its pilots and their responsibilities. However, this duplicates the same set of responsibilities already held by the PF and PNF actors in the current model, and may still fail to scale effectively to incidents with more than two aircraft, such as in the stack example. Alternatively, the core principles of this effect can be modelled simply by adding another responsibility that is held

by the Pilot Flying - 'Minimise effects on other aircraft', which encapsulates the various duties and obligations of the pilot to avoid unnecessary side effects from their actions.

Controller M also suggested adding an additional responsibility for pilots to explicitly acknowledge when they have completed their RA response and are now back under ATC control. Currently the model features the pilot informing ATC of an RA, and ATC acknowledging this, but it does not feature an acknowledgment of the end of the RA. This is a clear missing responsibility, and can be resolved by adding an 'Inform ATC of RA resolution' requiring the pilot(s)and the air traffic controller. Similarly, he also mentioned that air traffic controllers may modify their instructions to aircraft in order to limit the knock-on effects and will inform pilots of nearby aircraft so that they can try and reduce their knock-on effects. However, this type of behaviour falls out of the scope of this model, which is focused on the responsibilities immediately before, during and immediately after a TCAS alert (for a specific aircraft). These activities by an air traffic controller occur outside of that limited window with respect to the aircraft handling an RA as the aircraft is outside of ATC control at that point; they may advise *other* aircraft, but not those currently responding to an RA.

Interview Evaluation

The domain expert interviews were primarily designed in order to validate the responsibility model of TCAS, and partly to examine how well responsibility models and responsibility modelling can be understood and used to perform requirements analysis and elicitation. However, these two aims are tightly coupled and cannot easily be isolated from each other. Both interview subjects appeared to obtain a reasonable understanding of the responsibility modelling technique, and both also broadly agreed that the TCAS model was accurate (subject to various comments and criticisms). However, their ability to effectively analyse the TCAS responsibility model is dependent on their ability to understand the notation and core principles of responsibility modelling. If their understanding of responsibility modelling is weak then their ability to understand the specifics of the TCAS model is limited. Two distinct outcomes might produce the same results - the participants might have a good understanding of responsibility modelling and the TCAS model accurately represents their experience; alternatively, they may have a limited understanding of responsibility modelling, which makes them unable to detect subtle flaws in a TCAS model that is only average.

The positive results from the interviews allow us to rule out the least desirable outcomes - it is clear that at least a basic understanding of responsibility modelling can be obtained, and that the TCAS model does not feature significant amounts of serious errors. Participants were able to understand the notation enough to be able to identify some errors in the modelling of the problem domain, but did not find fault with most of the model. It is more difficult to assert from this that responsibility modelling is very easily understandable or that the TCAS model

is extremely accurate, as this requires separating out two variables from a single data point.

Completely separating the quality of the model from the understanding of the technique is an inherently difficult problem. Changing the first variable (the modelling notation / methodology) is impractical, as using a different modelling notation always requires using different models. With the choice of modelling technique fixed, changing the model offers some ability to examine the understandability of the technique independently by for example introducing deliberate flaws into a model to see if experts can detect them. Doing so eliminates the opportunity to study the quality of the model (as it has been made deliberately flawed) but could provide useful results about the understandability of the technique. However, unless the model has already been evaluated and validated it may be difficult to determine false positives and negatives (an error believed to be due to a poor understanding of the technique may in fact arise because the model itself is wrong). Such a study should therefore be conducted only after an initial validation has provided confidence in the models to be used.

7.8 Comparison to Similar Studies

Leveson [108] presents a socio-technical analysis of TCAS safety using the STAMP methodology [109], which takes a system-theoretic approach to safety analysis. For their analysis of TCAS they consider the potential contribution of TCAS to causing 'near midair collisions' (a violation of minimum separations), and do not consider secondary interactions of TCAS (such as confusing instructions causing the pilots to lose control of the aircraft). Their analysis begins by the construction of a process control loop for TCAS, as shown in Figure 7.6.

Following the STAMP methodology, potentially inadequate control actions are considered, which produces eight types of control failures (four each for TCAS itself and for the pilots). For TCAS these control failures represent failures of RA generation (such as generating an RA that actually reduces separation, or generating an RA too late to avoid an incident), while for the pilots they represent inadequate responses to the RAs (such as ignoring the RA, or not correctly following its instructions). Potential failures were analysed by using the SpecTRM (Specification Tools and Requirements Methodology) [110] executable model language. This requires constructing a model of the inputs and logic of both the pilots and TCAS - such as specifying the sensor information obtained by TCAS, and the pilots' mental model of their aircraft and nearby airspace. However, Leveson does not actually present the model in the case study [108]; given the scale and complexity of the system a complete model may not have been constructed.

An informal analysis then identifies several potential causes of vulnerabilities. For example, initialising TCAS after a plane is airborne will leave it non-functional until it obtains a first altitude reading, but pilots may assume that TCAS is functional as soon as it is powered up.

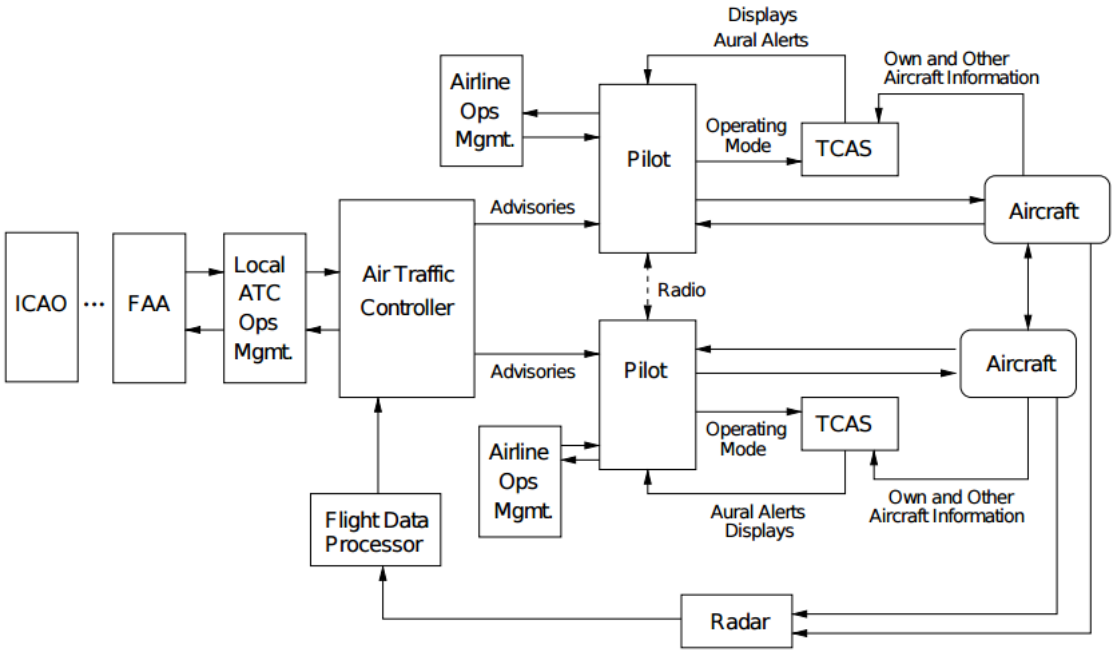


Figure 7.6: STAMP process control loop for TCAS (from [111])
Reproduced by permission of Nancy Leveson

Input sensors may produce uncalibrated or inaccurate information (such as the difference between measured altitude and true altitude), while time lags in processing may lead to false results (such as using round-trip time to determine the distance to another aircraft). They also note the importance of accounting for degraded operation over time, but do not provide any TCAS-related examples. Their work does not provide a formal or structured analysis of the vulnerabilities in TCAS. Their paper's main purpose is to propose and describe a new technique using TCAS as an example, rather than to perform a comprehensive case study of TCAS.

Felici [56] also consider a socio-technical safety analysis of TCAS as part of a study of air traffic management. They construct a SHEL [52] model of the Uberlingen mid-air collision, which is used to identify the information and timing of exchanges between pilots and ATC. They note that this incident arose as each participant had only partial knowledge of the full situation, and hence each actor had a different perception of risk. They go on to propose an evolutionary framework for modelling such complex interactions, but do not apply this framework to TCAS. Their SHEL model is not used to perform a comprehensive analysis, and is instead used to demonstrate its use as part of a broader technique.

Other approaches have also been used to model TCAS. Filipe et al. [58] create a model of interactions during the Uberlingen incident using timed automata, but this is not used to identify any vulnerabilities or make safety claims; the study is used to demonstrate the technique rather than analyse the incident. Jun et al. [96] provides a validation of the TCAS

decision logic via a coloured Petri net formalism, but this type of approach cannot be used to validate the interactions between TCAS, pilots and ATC. Similarly, Livadas et al. [113] provides a model of TCAS for analysing situations involving 'well-behaved aircraft', while Gotlieb [69] used constraint programming to assert safety properties of a fragment of TCAS implementation code. Numerous similar analyses have been published, but formal methods studies do not attempt to incorporate the socio-technical nature of TCAS operation. Instead, they focus on proving the correctness or effectiveness of TCAS's inputs and processing logic or its implementation. Potentially serious problems of this nature existed in previous versions of TCAS, such when issuing reversal RAs.

Studies have also been performed to examine the relationship between TCAS, pilots and ATC. Pritchett et al. [146] present a experimental user study of pilots using a flight simulator to observe and analyse professional pilot's reactions to TCAS TAs and RAs. They identify several interesting conclusions, including that pilots' self-reporting of ATC interactions is not reliable, and that there is a strong correlation between pilots' use of the TCAS traffic screen and their compliance with RAs.

In comparison to these techniques formalised responsibility modelling strikes a balance between modelling complexity and analytical power. High-level modelling techniques such as STAMP are used to construct models and informally identify potential vulnerabilities; this identification is not exhaustive, but locates some points of concern. In contrast, more complex modelling approaches such as timed automata can model a specific subset of the domain in rigorous detail, but at the cost of not being able to consider the full socio-technical nature of the domain. Formalised responsibility modelling provides a system-wide model of the domain, while providing more structured analysis than other socio-technical approaches. However, specialist modelling approaches may be still be needed to validate a higher-level approach, or to provide specific detail in areas of high concern.

7.9 Conclusion

This chapter modelled and analysed the TCAS collision avoidance system using responsibility modelling and evaluated the results of this technique with domain experts. Modelling TCAS revealed several ambiguities in the formal definition of the system between different sections of the same international standard and identified some features that are not in use, but the overall TCAS structure was analysed and found to have no significant failings.

Interviews with domain experts evaluated the accuracy and correctness of the TCAS responsibility model, as well as indirectly evaluating the understandability of the technique itself. These interviews revealed that our model was mainly accurate. Differences between the experts' experience and the model tended to arise in the original source materials used to

construct the model or the limited scope of the modelled system rather than due to oversights or inaccuracies in the modelling process. The experts were also satisfied they had achieved a satisfactory level of understanding. Certain aspects of the notation required some additional explanation (most notably the distinction between required and holds relations) but participants appeared able to quickly the grasp the general structure and semantics of responsibility models.

Applying responsibility modelling to the domain of TCAS proved effective in eliciting information from domain experts and moderately effective in identifying vulnerabilities, but also exposed several shortcomings in the current state of the technique. In particular, the effectiveness of several forms of analysis was challenged by the results of the interviews.

Overload analysis was shown to be of less use when applied to technical actors, as these technical systems are usually designed to meet specific performance levels, and hence are not at risk of overload when operating within a system as defined. Overload in technical actors occurs due to the violation of these performance levels (such as due to increased transaction levels, excessive numbers of users, etc.), while responsibility modelling identifies overload as occurring due to actors having to handle an excessive number of distinct responsibilities. As a result, overload analysis generates false positives for technical actors with large numbers of responsibilities, while not considering potential performance problems at all.

The complexity of the role and rank structures within the flight deck also highlighted some limitations of responsibility modelling. Actors in responsibility modelling are specific roles, and individual real-world individuals or organisations can fulfil multiple roles in a responsibility model. Responsibility modelling also provides techniques for considering the potential implications of this, such as using role analysis to identify potential conflicts of interests in such role allocations. However, this does not apply effectively to cases where there are additional meaningful interactions between actors that are not based on their roles - such as the rank/status dynamic between Captains and First Officers. In many cases this is captured by the use of distinct roles (for example the roles and responsibilities of a Private and a Captain in the army are quite distinct), but flight crew provide an interesting example where role-swapping between Pilot Flying and Pilot Not Flying occurs regardless of the difference in rank between the individuals.

The interviews also revealed an expanding use of TCAS outside the scope set for this model. TCAS was primarily designed as a system for preventing mid-air collisions by issuing warnings and advisories against short-term collision threats; detection and prevention of potential collisions in general would be provided by air traffic control measures. However, Captain A described significant use of TCAS not as a last-minute warning system, but as a tool used proactively to track nearby aircraft and improve pilots' awareness, allowing them to plan their actions better and avert potential incidents before they could occur. This behaviour

was out of scope for this particular study, but would be an interesting study of emergent behaviour in socio-technical systems and whether the technical aspects of these systems adapt effectively to the new uses that they are put to.

This chapter has demonstrated the use of formalised responsibility modelling in a new problem domain, and has shown that responsibility modelling can be used effectively to elicit comments and feedback from domain experts. Additionally, it has shown that formalised responsibility modelling's analysis results broadly correspond to known problems in the domain as identified by domain experts, and that the technique can be understood by domain participants with minimal explanation.

Chapter 8

User Study - Modellers' Application of Responsibility Modelling

8.1 Introduction

In previous chapters the use of responsibility modelling to model and analyse complex socio-technical systems has been demonstrated, and in doing so we have shown that responsibility modelling can effectively capture the behaviour of complex systems and produce accurate (Chapter 6) and relevant (Chapter 7) analysis results. However, there is an important limitation. All studies were carried out by the developer of the notation. This introduces two potential threats to the validity of the results. Firstly, the results may represent the skill or domain knowledge of the single modeller, rather than any more general strengths of responsibility modelling. Secondly, responsibility modelling may well be an effective technique when applied by an expert but be ineffective when applied by other, non-expert modellers. This tight coupling of technique development and modelling practice is not uncommon in socio-technical modelling. For example, all previous case studies of responsibility modelling were performed by developers or co-developers of the original notation. Baxter and Sommerville [11] note that 'In general, those who developed a method have had most success in applying it.'

Given this uncertainty around the wider applicability of responsibility modelling we seek to obtain more information by means of an empirical study. In order to investigate the applicability of the technique it is necessary to understand whether responsibility modelling is usable outside the community of expert methodology developers. This chapter describes a laboratory experiment in which non-expert modellers are tasked with preparing a responsibility model of a familiar domain - a meeting scheduler. Participants were first introduced to the technique with a short tutorial, before constructing a model from a short specification in their own time. Participants were then interviewed to understanding their experiences using the

technique; their models were also analysed, as was the correspondence of their models to their own views of the system.

The primary research question for this study is 'Can responsibility modelling be effectively applied by non-expert modellers?', which is then refined as follows:

Q1 - Can the responsibility modelling technique be understood and applied by non-expert modellers?

Q2 - Can non-expert modellers use responsibility modelling to produce models that represent their view of a system?

Q1 tests the methodology itself - are the constructs, concepts and notation of responsibility modelling understandable, and can modellers construct responsibility models that are syntactically correct? If the technique is too complex, this could lead to confusion amongst modellers and the production of syntactically invalid models. Q2 then examines whether the produced models actually represent the modellers' intended view of the system. It is possible that the complexity of the domain or inadequacies of the modelling constructs means that modellers cannot construct the model they wish to in responsibility modelling, even if they fully understand the technique.

Additionally, the qualitative nature of this study allows the examination of other issues that may occur during modelling or that are identified by participants, such as their modelling strategy, their understanding of specific responsibility modelling concepts and their impressions of the effort required. This is in contrast to previous studies of system modelling that have primarily relied on quantitative assessment (e.g. [122] [180]).

This study does not seek to examine if the models produced by participants are accurate representations of the problem domain. The level of accuracy is subject to a number of factors, including the domain experience of the modeller and the different information sources used. Additionally, it can often be difficult to define good metrics for the 'accuracy' of socio-technical models in all but the most trivial examples. Instead, this study investigates whether the models produced correspond to the modellers' own internal view of the system, and to understand why modellers produced their models in that way.

The remainder of this chapter is structured as follows. Section 8.2 examines previous studies of socio-technical modelling and requirements analysis techniques that used inexperienced participants to assess the understandability and usability of modelling techniques. Section 8.3 describes the experimental design of this study. Section 8.4 presents the results of the study, providing detailed evaluations of each participant's experience and displaying their models. Section 8.5 analyses the results of the study, discussing common issues and identifying participants' challenges in using the technique. Section 8.6 discusses potential threats to the validity of the study. Finally, Section 8.7 provides an overview of the study as a whole.

8.2 Similar Studies

Matulevičius and Heymans [122] reported an experiment to compare i* and KAOS, by dividing a graduate Computing Science class into two groups and allocating them a technique. These groups produced models in their assigned technique, and then evaluated both the technique and their models according to questionnaires based on a semiotic quality framework. The framework related the knowledge and interpretation of social and technical actors and the relationships between the domain, model and modelling language to produce eight quality types [103]. Students were first given a one sentence problem description, and time to elicit more detailed requirements from users. Student groups then constructed a model of the problem domain using either i* or KAOS (two teams per technique) and then evaluated both the methodology they had used and a model produced by another team using the other methodology, using a set of ordinal questions mapped to semiotic quality types. Their results comparing i* and KAOS were inconclusive, partly due to the small sample size, but they identify a number of important qualities for modelling languages.

The semiotic quality framework was also used in a more elaborate study by Moody et al. [127], which evaluated the quality of Entity-Relationship (ER) models while also assessing the framework itself. In this experiment participants were given several lectures explaining the framework, the ER notation and the evaluation exercise, and then assigned one of twenty case studies for which to produce a model within two weeks. Once these models were submitted, participants then made double-blind reviews of models using the quality framework. Finally, participants used a mixed-question survey to review the framework. The full experiment was carried out as part of a university course, using 192 students. Results were assessed using a range of quantitative and qualitative methods. These results showed an inconsistency between qualitative and quantitative methods, but validated the quality framework overall.

A study by Strohmaier et al. [180] investigated the use of patterns to improve i* modelling. This study consisted of creating abstract i* models for specific types of system (similar to architectural patterns) and then using these models as a building block towards modelling real systems. In Part A of the experiment students created an i* model for a collaborative working technology; in Part B they created a pattern for a different technology, and in Part C they attempted to apply a pattern produced by a different student. The sample size was very small (five students) and evaluations was conducted using a simple three question discussion, where the participants were asked which of the three parts was most difficult, were asked to rate their confidence on the correctness and completeness of the models and were asked which of the Part A and Part C models they found easiest to understand. These results are then discussed qualitatively, and Strohmaier et al. [180] conclude that patterns did not improve the quality of system models.

Maiden and Sutcliffe [118] perform a similar empirical study to assess their AIR (Advisor

for Intelligent Reuse) tool - essentially a tool that provides abstract domain models to aid in requirements engineering. Participants used this tool to analyse requirements problems, and then select between domain models. 'Correctness' of the choices was tested by examining the defined facts of each domain, which were then assessed for statistical significance with possible factors. Questionnaires were also used to gather additional information, but it is unclear what exactly was measured. Additionally, the participants' models themselves are used to understand their correctness scores. The sample size was 14, consisting of undergraduate and postgraduate students, as well as a single academic. Their study found that inexperienced modellers were unable to obtain the expected benefits of the reuse tool.

8.3 Experimental Design

In order to examine the use of responsibility modelling by non-expert modellers we conducted a qualitative, exploratory study where participants constructed a small (compared to those in Chapters 6 and 7) responsibility model. The construction of the responsibility model (after a brief introduction to the technique) allows examination of the first research question, investigating whether participants understood the technique and were able to apply it. After constructing the model participants were interviewed about the model they had constructed.

Our participants were drawn from postgraduate computing science students. This primarily represents convenience sampling, as such participants can be obtained relatively quickly and without the logistics of managing external participants. As an initial exploratory study it is unclear that any particular target sample should be aimed for, other than those who would make use of systems modelling techniques. Computing science students often (although not universally) make use of these techniques, although they are clearly less experienced than modellers in academia or industry.

This study consists of four parts:

1. Participants are introduced to the responsibility modelling technique
2. Participants produce a responsibility model of a specific domain
3. Each participant is interviewed about their model and their experience of using the technique
4. Qualitative evaluation of interview results and comparisons between models

Initially, participants were given an introduction to the responsibility modelling notation and technique in tutorial format, including descriptions of all entities and relations and a worked example. This tutorial introduced the general motivation for responsibility modelling,

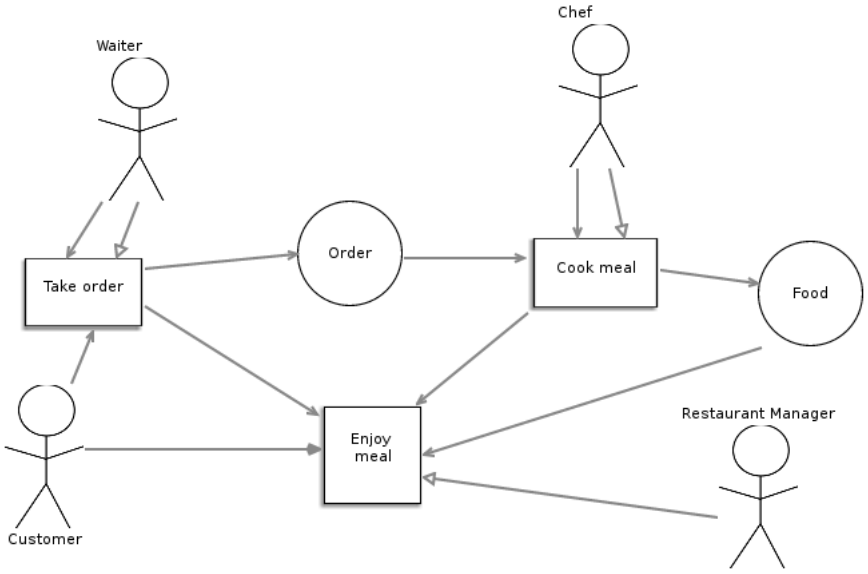


Figure 8.1: Tutorial example of a responsibility model

the three entity and five relation types and described the graphical notation. This was introduced by showing and describing a responsibility model of ordering and eating a meal in a restaurant (Figure 8.1). Additionally it discussed some unusual characteristics of responsibility modelling (the complexity of holding a responsibility without being required by it, and the nature of time) and described a methodology for generating responsibility models from source materials. This tutorial does not describe all features of formalised responsibility modelling and omits topics such as the constraint language and automatic analysis, in order to reduce the difficulty of participants in understanding the tutorial. Additionally, participants were given a very brief introduction to the RESME responsibility modelling toolkit, which they were recommended to use to construct their model. Participants also completed a short questionnaire which asked about their previous experience with modelling techniques, in order to explore whether past modelling experience with other techniques influenced their use of responsibility modelling.

After being introduced to the technique participants were asked to construct a responsibility model of a specific case study. For the purposes of this experiment the ideal case study should describe a socio-technical system of moderate complexity (providing enough detail to make the modelling task non-trivial, but not so much as to discourage novice modellers) without using excessive domain-specific language and terminology. Additionally, there are advantages to using a case study not specifically designed for the experiment - this limits the effect of experimenter bias, which might lead to the introduction of features particularly suited to the modelling notation being studied. This case study chosen is the meeting scheduler problem as initially proposed and modelled in KAOS by van Lamsweerde et al. [189] and also modelled in i* by Yu [202].

The Meeting Scheduler System is intended to support the organisation of meetings. Meetings are arranged in the following way. A meeting initiator asks potential attendees to list dates within some range that they would prefer to attend a meeting (the preference set) and dates where they cannot attend (the exclusion set). Additional preferences may be expressed by attendees, such as requirements for special equipment or choices about the meeting location. A proposed meeting date should belong to as many preference sets as possible, and not be a member of the exclusion sets; if this cannot be achieved, a date conflict has occurred. Our version of the problem definition is slightly different from the canonical version [188] as we removed references in the introduction to the definition being deliberately vague, and removed the list of potential extensions at the end of the specification in order to limit the scope of the experiment.

The definition contains a number of non-functional requirements (such as the support for concurrency) and secondary features (such as privacy rules), which can be considered auxiliary to the main specification of the system. This means that the exact set of features included in any particular model of the system can vary quite widely (for example, neither Yu's i* model or van Lamsweerde et al.'s KAOS model cover all of the requirements). This is helpful to the study's research goals, as it introduces variance into what the participants intend to model.

All these experimental materials are available online in a GitHub repository [161]. Additionally, the case study definition and responsibility modelling tutorial are presented in Appendix A.

Participants were asked to construct a responsibility model of the meeting scheduler problem in their own time, ideally using the RESME toolkit. The participants were instructed to construct their model unsupervised at a time and location of their choice, but were advised not to spend more than a couple of hours performing this task. Participants contacted the researcher after they had completed their model, and were then called back for a debriefing interview.

These were semi-structured, individual interviews using a series of probes. Initially, participants were asked to describe their model, providing their own interpretation of the responsibility model they had produced. This provides a baseline for assessing if the model accurately represents the participant's intention when they were constructing the model. After this description questions were asked about elements of their model. These included questions about the notation (such as if they had not used a particular feature of the notation, or if their model did not conform to the notation described in the tutorial) and questions about the model structure (such as if large areas appeared to be missing, or if it was difficult to follow the design of the model).

After initially discussing the model, participants were then asked about their experiences of responsibility modelling. Firstly they were allowed to respond to this unprompted, raising

their own issues. Participants were then prompted to describe the parts of the technique they found easiest and hardest to understand or use in order to gather additional information. At this stage any deviations from the standard notation in their model were examined in more detail. After this, participants were then asked if there were any parts of the system they could not model in responsibility modelling, and to explain why if so.

Finally, the participant's model and the original system definition were compared. Participants were asked about any significant differences between the model and the definition to investigate why these had occurred, aiming to distinguish between differences that had arisen due to the use of responsibility modelling (e.g. because responsibility modelling lacked the ability to represent a particular area or function) and those that were due to other factors (such as lack of time to model, or the participant not understanding a certain section).

8.4 Results

All participants were postgraduate students within the School of Computing Science at the University of Glasgow. In total, seven participants took part in the study; all those who started the study continued to completion.

8.4.1 Participant A

Participant A has a background in security research; their experience of modelling techniques was limited to studies of entity-relationship diagrams [28] and the Unified Modelling Language [152] as an undergraduate. They had not performed any substantial projects involving systems modelling before, and had not previously encountered responsibility modelling.

Participant A's responsibility model of the meeting scheduler is shown in Figure 8.2. To initially construct their model they used a simple variation of the elicitation technique described in the tutorial, underlining key words and phrases in the problem definition and then classifying them as the appropriate modelling concept.

The overall structure of this model is broadly consistent with the responsibility modelling notation presented in the tutorial - initial responsibilities such as 'Get Meeting Info' produce resources that are then used to finalise the choice of date and meeting, which allows the meeting to be scheduled and take place. The purpose of the Meeting Scheduler Definition is to define a computerised meeting scheduler system that 'supports the organisation of meetings' by automating communication and reducing overhead. Participant A's model makes no reference to such a computerised scheduling system - many of the responsibilities and decisions within the model could be made by a scheduling system, but they could equally describe a purely manual system. However, the definition document describes the process in

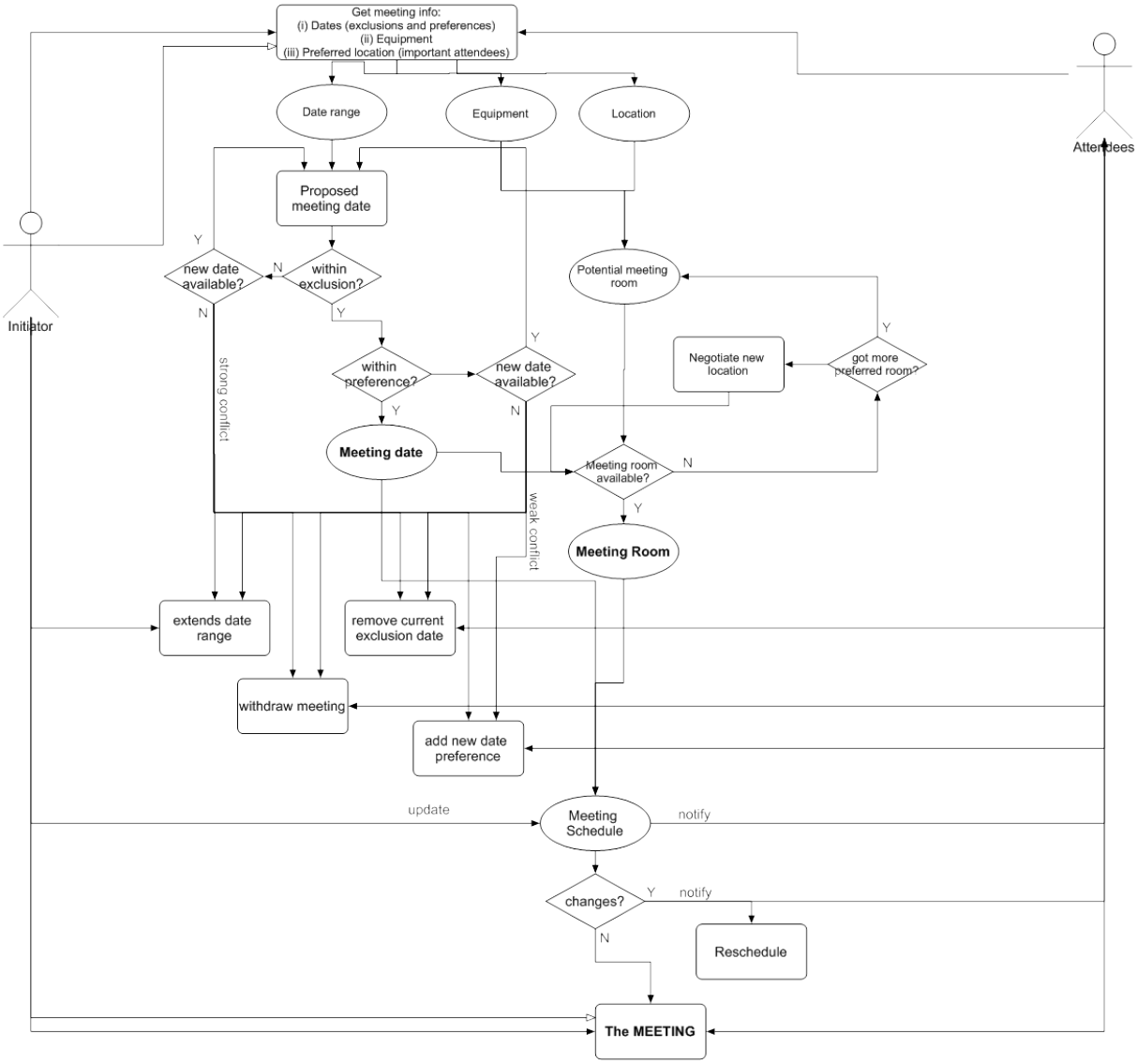


Figure 8.2: Participant A's responsibility model

ambiguous terms and mentions a computerised system for the first time half-way through the document, so overlooking it is understandable.

However, there is one obvious deviation from the standard responsibility modelling notation - the diamond shape choice or decision objects (similar to those in the Unified Modelling Language). These are used to represent the various conditions under which the meeting date or room is determined, as the problem definition clearly describes the process for identifying and resolving conflicts. The use of this extra notation is in conflict with the style of responsibility modelling used in previous studies. Responsibility models have generally avoided describing choices in detail, choosing instead to abstract away this level of detail. For example, the 'within exclusion' and 'within preference' decision objects might be abstracted into a 'Choose meeting date' responsibility. However, this is not necessarily an inherent characteristic of the methodology, and instead reflects the preferences of certain modellers. Indeed, the desire to explicitly represent complex scenarios involving multiple possible options led to the development of a constraint language for responsibility modelling, as described in Chapter 4. The constraint language allows for the specification of alternative options for discharging a responsibility or producing a resource. For example, Participant A's decision chain for the availability of a meeting room could be expressed as:

MeetingRoom: Room Available || NegotiateNewLocation Discharged & GetNewPreferences Discharged.

Their use of the UML-inspired flowchart notation is effectively a less formal graphical representation of the constraint logic already featured in responsibility modelling, but left out of the participants' tutorial for reasons of brevity. Participant A explained their use of these flowchart elements as being inspired by the UML, explaining that human factors mean that a system must have several options to be considered. They also noted that they had used the UML notation because they "did not really know the [responsibility] model [notation] well"; they felt the behaviour was important to capture, but had not seen any suitable notation to capture it with.

In contrast, their model demonstrates effective use of the differences between the holds and requiredActor actor-responsibility relations. All relations between actors and responsibilities use the standard requiredActor notation, with two responsibilities also being held by the Initiator - the initial 'Get Meeting Info' responsibility and the final 'The Meeting' responsibility. In both of these cases the Initiator holds the responsibility and the Attendees don't, indicating the authority and duty of the meeting's original organiser. They also occasionally used labels on relations to describe the nature of the interaction between entities: for example, both the Initiator and the Attendees are linked to the Meeting Schedule - the Initiator is required to *update* it, while the Attendees must be *notified*.

When asked to describe their model their verbal description of the system corresponded well

to the responsibility model that they had produced. They described in some detail the process required for a meeting initiator to set and then finalise a potential meeting date, given the constraints placed upon it. This process is clearly expressed in the flowchart-type notation used in the relevant section of the model. Similarly, they provided a clear explanation of the process for obtaining a suitable and available meeting room, which is also expressed through the flowchart notation.

Participant A's experience of responsibility modelling was generally straightforward. They found the hardest part of the exercise was "coming up with the requirements"; understanding the problem specification and the logic within. Once they had understood the specification "it was easy to follow" how the processes and logics could be captured using responsibility modelling, by considering each element to determine if it should be modelled by a resource or a responsibility. This modelling process demonstrated a learning curve, with the first objects being difficult, but becoming easier once they had grasped the basics. In particular, understanding and then modelling the complex logic for determining the meeting time and location (by considering the options and constraints) was identified as the hardest area of the task to complete, with understanding the specification the main issue - "I'm OK with translating this once I've got the idea".

When asked about parts of the specification that they were not able to model they mentioned the parallel aspects of the system, such as 'the meeting scheduler must in general handle several meeting requests in *parallel*'. They expressed uncertainty over the meaning of this phrase - did this refer to the system being run repeatedly, or the need to handle multiple meetings at once? Would this would affect the rest of the system, such as needing to check if participants were attending meetings simultaneously?

Their responsibility model did not feature separate actors for the different types of participants (the definition mentions participants, active participants and important participants), instead using a single, complex responsibility to model the different preferences expressed by the participants. This makes that responsibility more complex, but Participant A noted that this eliminates the need to specify how how the status of a participant is determined - "the more complex thing is how you determine which one is important". The model also lacks most of the non-functional requirements expressed on the final page of the definition - Participant A noted that these were subjective, and it was not clear how they could be modelled in the notation. This is a known shortcoming of responsibility modelling, and occurs to some extent in most socio-technical modelling techniques - if processes are abstractly modelled then it is difficult to specify non-functional requirements, as these may vary depending on implementation.

Overall Participant A was able to construct a reasonable responsibility model that matched their understanding of the system, but was only able to do this by introducing their own

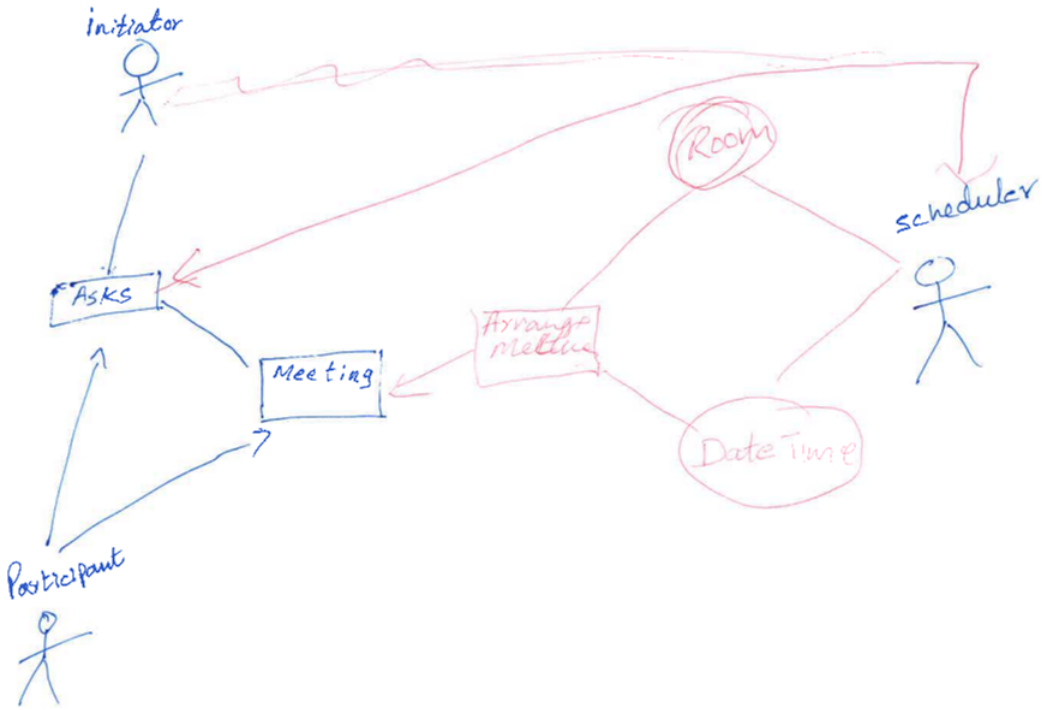


Figure 8.3: Participant B's responsibility model

notation to capture complex choice-based behaviour. The use of the decision objects allowed them to represent all parts of the system definition that they wished to, with the exception of the non-functional requirements. The most demanding part of the modelling process appeared to be understanding the problem domain and the definition, rather than the responsibility modelling itself; however, their model features a number of small deviations from the standard responsibility modelling notation. This may reflect limitations in the tutorial, or producing a fully compliant model may require additional effort or concentration.

8.4.2 Participant B

Participant B has previous experience in requirements engineering (using methods such as misuse cases and TROPOS [20]) and had made use of these techniques in a study comparing formal and informal requirements engineering techniques. They also have experience with the use of safety cases and large-scale system models, but had not previously read about or used responsibility modelling.

Participant B's responsibility model of the meeting scheduler is shown in Figure 8.3. The hand-drawn nature of this model leads to a notation that is often inconsistent with the notation presented in the tutorial. For example, several relations are presented without directionality (e.g. the relation between 'Asks' and 'Meeting', or between 'Date and Time' and 'Arrange Meeting'. This makes the model difficult to interpret, as both potential relations implied by

this link (e.g. that arranging a meeting requires a date and time, or that the act of arranging a meeting produces a date and time) are potentially valid. However, Participant B was able to provide a description of these relations when asked; the lack of directionality appears to indicate difficulties constructing the model or understanding the notation, rather than a lack of understanding of the relationship in the domain.

Additionally, they use only one form of relation between actors and responsibilities, apparently disregarding the 'holds' relation. When queried, they simply stated that they "missed that", and asked for a step-by-step process for modelling relations. The relation between the Scheduler and 'Asks' also demonstrates a non-standard double-headed arrow, implying a bi-directional relationship; Participant B explained this relationship as the Scheduler using the information provided by the 'Asks' responsibility, but did not comment on the bi-directionality - "[the] Scheduler uses these resources; it's the Scheduler's responsibility to do this.". However, while the relations are mainly non-standard the entities all correspond directly to those presented in the responsibility modelling tutorial.

Participant B seemed unclear about many of the relations when asked to explain their model. For example, resources are linked directly to actors (such as 'Scheduler' and 'Date Time') and they were not able to provide a clear explanation of how these resources were produced.

They also expressed several problems with the task, unprompted. They felt the system definition contained significant technical jargon, limiting understandability. They also wished that they had a step-by-step process for constructing a responsibility model, such as identifying entities, then identifying relations and so on - "it would be very nice if we had steps [for making the model]". Both of these are useful observations - parsing complex domain-specific language is an integral part of many modelling methods.

In contrast to many other participants, Participant B explicitly included an actor representing the computerised scheduling system. They stated that "it is very clear" from the tutorial that actors could be computer systems, but suggested that the use of a stick-figure notation for actors might confuse others.

Participant B also suggested improvements to the modelling notation. They argued that relationships would be much clearer if labelled with short descriptions - "the arrows need labelling", which would make it possible to understand a model visually without having to read a description to obtain the full detail. For example, they considered the relation between the Manager and the 'Enjoy Meal' responsibility in the tutorial to be unclear (as show in Figure 8.1), thinking that the manager enjoyed the meal with the customer. When asked about areas of the notation that were particularly easy or particularly difficult to understand they again referred to the use of labels or similar additions to the notation, with the intent of making the diagram easier to understand.

When asked about parts of the specification that they were not able to model they mentioned

the “functionality behind” arranging the meeting, the “process or implementation” details involved. They suggested a pseudocode type notation for describing algorithms. They also discussed the possibility of a deadlock situation, such as when no rooms met the required preferences; this presumably requires some preferences to be dropped, but this is not included in the model. Both of these issues are related to the model’s level of abstraction - responsibility modelling would normally be used to abstract over certain implementation details, and the core concept of a responsibility is not suitable for low-level implementation details, where a process-based model may be more suited. Additionally, they mentioned the idea of “partial accountability”; they noted that accountability “is not zero or one”, such as in the case of the meeting scheduler, as some accountability should be placed on the participants for the preferences they express. Responsibility modelling does not provide a clear solution to this. Responsibilities are either held by actors or are not held, and there is no notation for indicating the strength of the relation.

In several areas their responsibility model was very abstract, such as all interactions between the scheduler and participants being covered by the one word responsibility ‘Asks’. However, they were able to given a detailed description of these interactions when prompted, suggesting that the level of detail was not due to a lack of understanding, but might be attributable to a difficulty converting into responsibility modelling notation, or perhaps purely a lack of time. Their responsibility model did not feature separate actors for the different types of participants, nor did it mention how they expressed different types of preferences. Participant B said that this was primarily “for simplicity”.

Overall, Participant B’s model was highly abstract, lacking the majority of details contained in the system definition. Their model also demonstrated an inconsistent application of relations, with a mix of non-directional, one-directional and bi-directional relationships indicated. This might be partially be down to the pressures of time the participant indicated, but the inconsistent notation suggests a difficulty in understanding responsibility modelling. This is supported by Participant B’s suggestions for improvements, which primarily focused on ways to make responsibility models easier to understand. Alternatively, the lack of detail may be caused by a feeling that responsibility modelling lacks the right concepts for representing detailed actions, as indicated by their comments on modelling details of processes.

8.4.3 Participant C

Participant C has previous experience with workflow and trust modelling, as well as a basic understanding of UML and ER modelling. They had previously worked on a project to implement workflow modelling in Python that could be ‘fuzzed’ - introducing random changes to the model to determine a system’s robustness. They had also read some early work

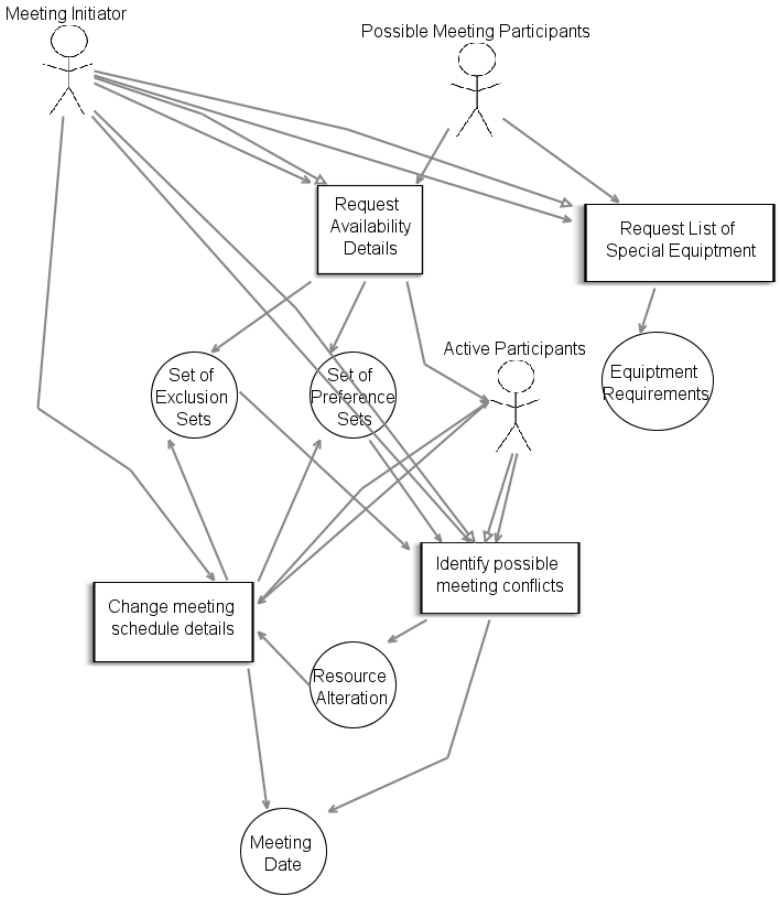


Figure 8.4: Participant C's responsibility model

on responsibility modelling (such as Sommerville [167]) but had never used the technique themselves.

Participant C's responsibility model of the meeting scheduler is shown in Figure 8.4. They constructed their model using the RESME toolkit, ensuring that it matches the standard notation described in the tutorial. However, they used one part of the notation that is supported by the RESME toolkit but was not described in the tutorial - the 'Produce Actor' relation, which was used to model that Active Participants are created by the 'Request Availability Details' responsibility. This is a logical relationship - participants are feasibly only active if they are available (the system definition does not define active participants), and shows that the participant was able to use features of responsibility modelling that had not been explicitly introduced.

Their model shows heavy use of resource intermediation to convey interactions and responses - for example, the responsibility 'Identify possible meeting conflicts' produces a resource 'Resource Alteration', which is then an input to the 'Change meeting schedule details' responsibility. It also demonstrates the coupled use of the 'holds' and 'Required Actor' relations; only once are these not used together - Possible Meeting Participants are only required by

'Request Availability Details', which both requires and is held by the Meeting Initiator.

Participant C noted that they were unable to find a suitable way to represent choices between different actions, which affected their construction of the model. They wanted to be able to model "an obligatory choice between an arbitrary set of actions", such as when identifying meeting conflicts. In this case they used a resource that was generated when there was a change of meeting schedules, which is then used to produce a new set of preferences and exclusions. This type of behaviour can be formalised in responsibility model using the constraint language, but this was not described in the tutorial presented to participants.

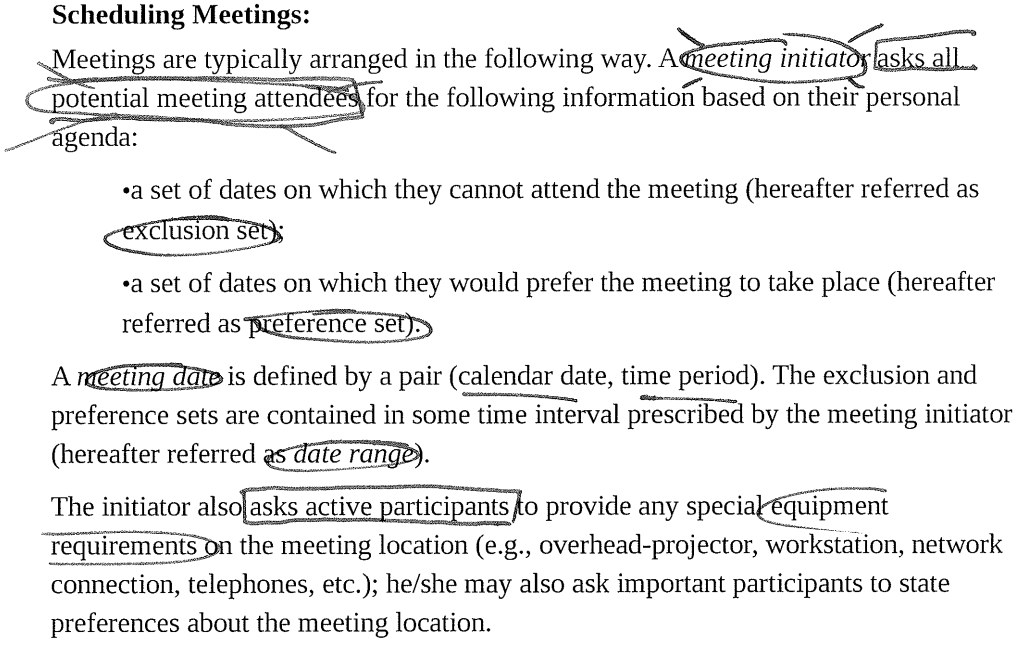
They also indicated that their model has an effective "arrow of time" - the system begins by the initiator requesting availability details, which produces participants that will take part in the meeting, which then produces a potential resource alteration, and so on. However, this process can also loop. Timing in responsibility modelling is a complex issue without explicit ordering, but inferred orderings like the 'arrow of time' occur in most models.

Participant C often indicated that they knew their model was incomplete, or that they were not sure if they had applied responsibility modelling correctly. For example, they were aware that the 'Equipment Requirements' resource was never used, but felt that "it didn't seem to be needed to be used", based on the system definition. They also noted a number of constraints in the definition that were not included in their model - primarily the non-functional requirements on the final page of the definition. They expressed this as a separation of concerns between the responsibilities that had to be modelled and the processes that were modelled; the constraints applied more to the processes, while the responsibility model naturally focuses on the responsibilities.

No part of their responsibility model explicitly features computerisation, as is implied by the system definition. The model is abstract enough that the level of automation is left ambiguous; there may be computer systems implementing some of the tasks described, but they are not part of the responsibility structure. Participant C noted that they had thought about this, but felt their model should not be implementation specific, and hence would not contain explicit references to a computer system.

To initially construct the model they had used a similar technique of entity identification to that described in the tutorial. They annotated the system definition document to identify actors, responsibilities and resources, as shown in Figure 8.5. They used different notations to identify each type of entity in the text - rectangles for responsibilities, ovals for resources and diagonally-cut ovals for actors.

Participant C found the graphical representation of responsibility to be easy to use - "I found the notation really, really easy to create and reason with". They did not identify any problematic areas of responsibility modelling, although they did note an initial learning curve. They spoke highly of the RESME toolkit, describing it as "very easy to use", although the



Scheduling Meetings:

Meetings are typically arranged in the following way. A *meeting initiator* asks all potential meeting attendees for the following information based on their personal agenda:

- a set of dates on which they cannot attend the meeting (hereafter referred as exclusion set);
- a set of dates on which they would prefer the meeting to take place (hereafter referred as preference set).

A *meeting date* is defined by a pair (calendar date, time period). The exclusion and preference sets are contained in some time interval prescribed by the meeting initiator (hereafter referred as *date range*).

The initiator also asks active participants to provide any special equipment requirements on the meeting location (e.g., overhead-projector, workstation, network connection, telephones, etc.); he/she may also ask important participants to state preferences about the meeting location.

Figure 8.5: Participant C's annotated system definition

installation process was off-putting. Initially they found adding elements into the model difficult, but "this got easy after about a minute". They had learnt to use only one of the two methods for adding entities to a model; without a tutorial on the toolkit itself they had to learn by trial and error. Their actual creation of the responsibility model only took about half an hour; the remainder of their time was spent understanding the definition and identifying the various entities. They did have problems attempting to find a satisfactory layout for the model; the toolkit's automatic layout was unsatisfactory, and the complexity of layout encouraged them to use fewer relations.

When asked about parts of the specification that they were not able to model they mentioned resources - their model used several examples of sets of sets (such as 'Set of Preference Sets'). They wanted to be able to model a "plurality of a resource", in that the participants each produce a set of preferences, which are aggregated by the scheduling system. This implies a one-to-one mapping between a set of preferences and a participant, and Participant C would have liked to show this mapping explicitly, which would add clarity to the model, and distinguish between singular and multiple actors (such as having only one Meeting Initiator, but multiple Active Participants). More generally, plurality of entities could be used to identify responsibilities that are repeated (such as identifying meeting conflicts). Some of this behaviour can be modelled using the constraint language, but the structure of responsibility modelling forces intermediation - for example, a resource cannot be produced by an actor; instead, the actor should hold a responsibility that produces the resource.

Their model does not feature the selection of a location for the meeting, despite obtaining

equipment requirements (which would be used to identify suitable rooms). They chose not to include this as they were not sure how it would fit into their model; on reflection, they felt that it could be added as another input to identifying meeting conflicts. More generally, the equipment requirements display several oddities: the responsibility requires possible participants, not active participants (implying that non-attendees could still express preferences) and the resulting 'Equipment Requirements' resource is never used. Participant C felt that this resource was not needed by any other part of the model; with hindsight they felt this could be used as another source of potential conflicts, such as if the room selected lacks the correct equipment.

Overall, Participant C constructed a substantial model that provided a detailed description of the system. Their fidelity to responsibility modelling was enforced by the use of the RESME toolkit for modelling, but they also showed effective use of relations, including those that had not been introduced in the tutorial. They effectively grasped the standard responsibility modelling approach to abstraction based on responsibilities rather than processes, possibly driven by their reading of similar work. Their main limitation was the inability to express more complex behaviour, such as complex relationships between actors and resources; this can largely be expressed through using the constraint language, but the participant was not introduced to this.

8.4.4 Participant D

Participant D has only limited experience with modelling techniques, having previously studied the basics of UML modelling, but not had applied it in any meaningful projects.

Participant D's responsibility model of the meeting scheduler is shown in Figure 8.6. This is a simple model consisting of just seven entities. Their model adheres tightly to the responsibility modelling notation defined in the tutorial. Their use of relations is clear, demonstrating logical use of the differentiation between 'holds' and 'required actor'. However, their naming of responsibilities is unclear - the responsibility 'Meeting' is short and unspecific, while 'Proposed Date' implies an (information) resource rather than being a responsibility. They were able to give substantial definitions of these responsibilities when asked, implying that the short names are part of their desire for simplicity.

Participant D emphasised simplicity when constructing their model - "I wanted to try and keep it as simple as possible". They were concerned that a complex model would be difficult to understand and would cause people to lose interest, but might still not be complex enough to describe all possible scenarios - "If you try to capture everything you get a very complex system". For example, they noted that choosing the date of the meeting might require multiple rounds of negotiation between different sets of participants, and so chose to abstract over

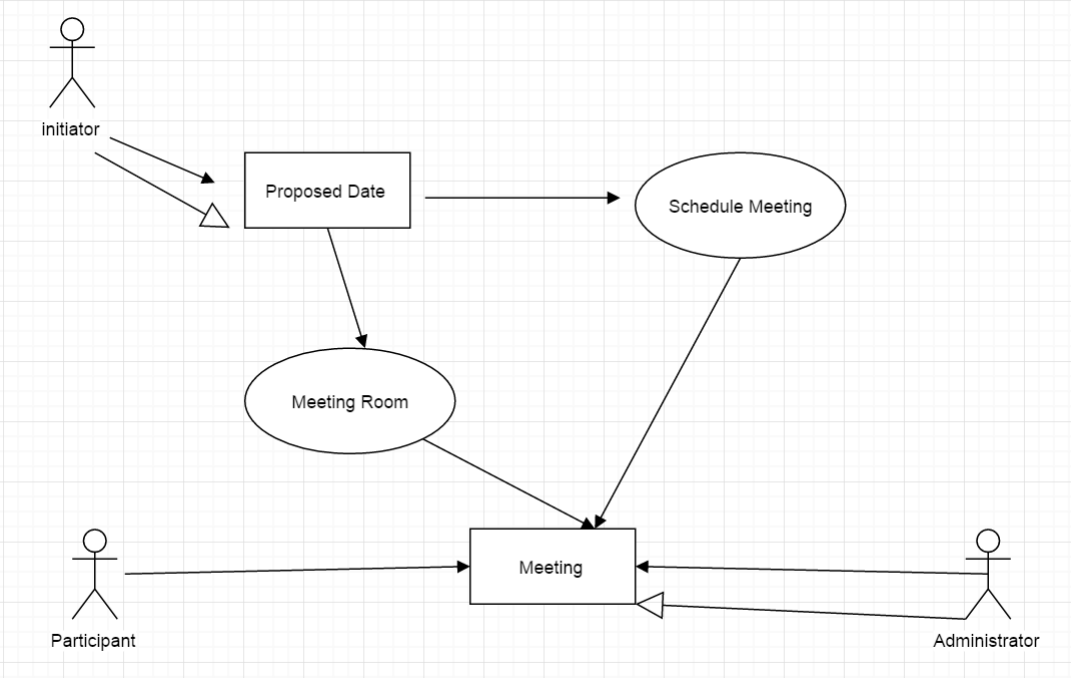


Figure 8.6: Participant D's responsibility model

this detail instead. They were aware of the possibility that more detail might be required in some cases - "if there is a need to, you can have other subsections". This simplicity extended to the names of responsibilities, which were extremely short. However, Participant D was able to provide an extensive description of the activities encapsulated in these responsibilities when queried; they clearly had a strong understanding of the system definition and how their model related to it, but chose simple descriptions to describe complex behaviour. Their description of the 'Proposed Date' responsibility identified a potential flaw - they described how participants would express their date preferences, but Participant was not a required actor for the responsibility. However, Participant D quickly identified this as an oversight when mentioned, and it is unclear why they did not include it originally.

They also introduced a new actor into the system - the Administrator, which does not directly correspond to any of the agents mentioned in the system definition. They felt that "there was too much work for the initiator" and that this addition would make the work of the initiator easier. They suggested various duties for the administrator, such as ensuring the meeting starts and ends on time and that the venue is ready and configured. The concern that the initiator has too many responsibilities is a core part of the system definition, which is intended to enable the design of a computerised scheduling system. However, the responsibilities indicated for the scheduler are quite different from those Participant D assigned to the Administrator; the scheduler is intended to automate the process of fixing a time, date and location, while the Administrator's responsibilities are actions that take place after the meeting has been scheduled.

Participant D began their modelling by studying the responsibility model tutorial. They then successfully installed the RESME toolkit and constructed an initial model - “I got everything set up and already created one [model]”; but then were unable to reload it due to Eclipse updates and resorted to a general-purpose drawing tool. When analysing the system definition they began by identifying actors, then identified the other entities. They deliberately constructed an initial model that was considerably larger than their final model (for example, it contained multiple types of participants) before simplifying and refining their model to produce the final result. They generally found responsibility modelling straightforward, and said they didn’t have any challenges understanding the technique. They also found understanding the problem definition easy; they spent most of their time using the tooling or revising and refining their initial model - “the problem definition was easy; it was just implementing it [the model] that was the challenge”.

While they did not find themselves unable to model any parts of the system, they did express some dissatisfaction with the use of resources. At times they would have preferred to link actors directly to resources, rather than having to intermediate using a resource. This use of responsibilities as intermediate elements is an important prerequisite to automatic analysis, but ‘syntactic sugar’ could enable visual shorthands while retaining the original underlying representation.

Compared to the system definition their model lacked many features, as they had sought to simplify and reduce complexity. They deliberately simplified parts of the system that had many options (such as choosing dates, or the types of participants) in order to make the model easier to understand. Similarly, they chose not to model the replanning process, as this process might take many rounds and was not guaranteed to produce a result - “you might have a continuous loop... but you might never have everybody” . They suggested the use of sub-models to provide more detail on individual responsibilities if that level of detail was required. They also discarded most of the non-functional requirements, feeling them to be overspecified and unnecessary for most cases of scheduling. This was also their justification for not making explicit references to a computerised scheduler; their model was intended to be abstract enough to cover most scenarios, and extra detail could be added for a specific circumstance - “I felt it was adaptable...”.

Overall, Participant D constructed a relatively simple, highly abstract model of the system, but this simplicity was an explicit modelling decision rather than due to any lack of understanding of the domain. Their application of responsibility modelling was close to the notation provided in the tutorial, but their desire for simplicity lead to deviations from the problem definition. They appeared confident in the use of the technique; differences in their model arose from their different perspective on and motivation for modelling, rather than any lack of knowledge.

8.4.5 Participant E

Participant E has extensive past experience in systems modelling, having worked with large consultancy firms on government IT projects. They have experience with UML, the Rational Unified Process [104] and the Archimate enterprise modelling language [82], but had not previously encountered responsibility modelling.

Participant E's responsibility model of the meeting scheduler is shown in Figure 8.7. They produced a complex model consisting of dozens of entities, arranged by type. Their model contains almost all concepts described in the system definition. Actors are comprehensively defined, including an explicit 'System' actor and the inclusion of a participant's representative. Similarly, an exhaustive listing of resources and responsibilities is provided, including both functional elements (such as preferences and actions) and non-functional system characteristics (such as concurrency and usability). There is significant overlap between the set of resources and the set of responsibilities, as many concepts appear twice - for example, there is a resource 'preference set' and a responsibility 'preference sets'.

The model shows substantial interaction between entities, but the relations take very specific forms. Actors are directly linked to both responsibilities and resources, and the 'holds' relation is not used. Resources and responsibilities are linked, but the substantial majority of links are from resources to responsibilities (production) with limited links from responsibilities to resources (consumption). Additionally, resource to resource links are used (such as 'special requirements' being linked to 'meeting room'), but there are no homogeneous links between responsibilities or between actors. This type of relation is a not a part of standard responsibility modelling; conceptual links between resources are common (such as special requirements influencing the choice of a meeting room), but these are intermediated through responsibilities.

Given the number of entities featuring in the model the structure is flat, with relatively few complex interdependencies. The most interdependent section of the model resolves around identifying strong and weak constraints - the constraints depend on the conflict date, which itself relies on the meeting room, which then relies on special requirements, which is ultimately reliant on the meeting initiator and attendees. However, many other entities are only involved in one relation, such as the non-functional system requirements that are not linked to any other features.

Participant E used "their usual habit" of identifying the "functionalities" of the system, which they modelled as resources, and then the actors. They then constructed the relationship between these functionalities and the actors. This suggests an uncertainty of the difference between responsibilities and resources; in other responsibility modelling studies system functionalities would be represented as responsibilities, with resources used to model their inputs and outputs. The distinction between functionalities, resources and responsibilities is not as clear in Participant E's model - many of what they describe as functionalities

Figure 8.7: Participant E's responsibility model

are pieces of information (such as ‘Preference Set’, ‘Special Requirements’ and ‘Proposed Meeting Date’), while their responsibilities include both functionalities (e.g. ‘Add new dates to preference set’) and more abstract concepts (e.g. ‘Domain theory’, ‘Conflict Resolution’).

They found it “very difficult” to determine the responsibilities. They noted that they were very familiar with analysing actors and system functionality from using other modelling techniques but that they had never considered the more abstract idea of responsibility, and suggested that guides and tutorials should emphasise the nature of responsibilities as this concept was less commonly known. In particular they were used to the idea of ‘sequence’ - “in our requirements we get the sequence of events”, while responsibility modelling uses a different temporal concept. After constructing a set of responsibilities they had further difficulty in deciding which actors would be responsible for which responsibilities. This choice of language implies that they considered the responsibility / accountability relation to be the main component of actor-responsibility notations, even though they only used the ‘required actor’ notation in their model. In contrast, they found it very easy to determine the responsibilities for the ‘System’ actor, which consist primarily of non-functional requirements - “I found it very easy to do this part. You know the clear-cut responsibilities”.

Participant E had particular difficulty in attempting to model the identification and resolution of date conflicts, as there was no method in responsibility modelling to express the “alternative flow”; that different actions might be taken depending on the state of the system, such as depending on whether a suitable date could be found. This type of behaviour can be at least partially modelled using the responsibility modelling constraint logic, but this was not included in the tutorial given to participants. They also noted the absence of sequencing in responsibility modelling, which does not have an explicit mechanism for timing.

They justified their lack of direct links between resources and responsibilities by describing how they are linked indirectly - for example, the ‘Add new dates to preference set’ responsibility is held by the ‘Meeting Initiator’ actor, which is linked to the ‘preference set’ resource - “I have shown the link between the actor and the responsibilities, but I’ve not shown the link between the resources and the responsibilities. It’s an indirect link”. This does not correspond to the conventional structure of responsibility modelling (where direct actor-resource relations do not exist, and responsibility-resource relations are standard) but may provide a simpler representation given the large number of entities in their model. They also addressed the similarities between some resources and some responsibilities (such as the two occurrences of ‘preference set’) by noting the convention that verbs in free text represent actions, while nouns represent resources. Therefore the term ‘preference set’ can refer to both the information itself (a resource) and the use of the preference set as part of another process (a responsibility).

Overall, Participant E constructed a very detailed model that provided an extensive representation of the system definition but did not correspond to the responsibility modelling

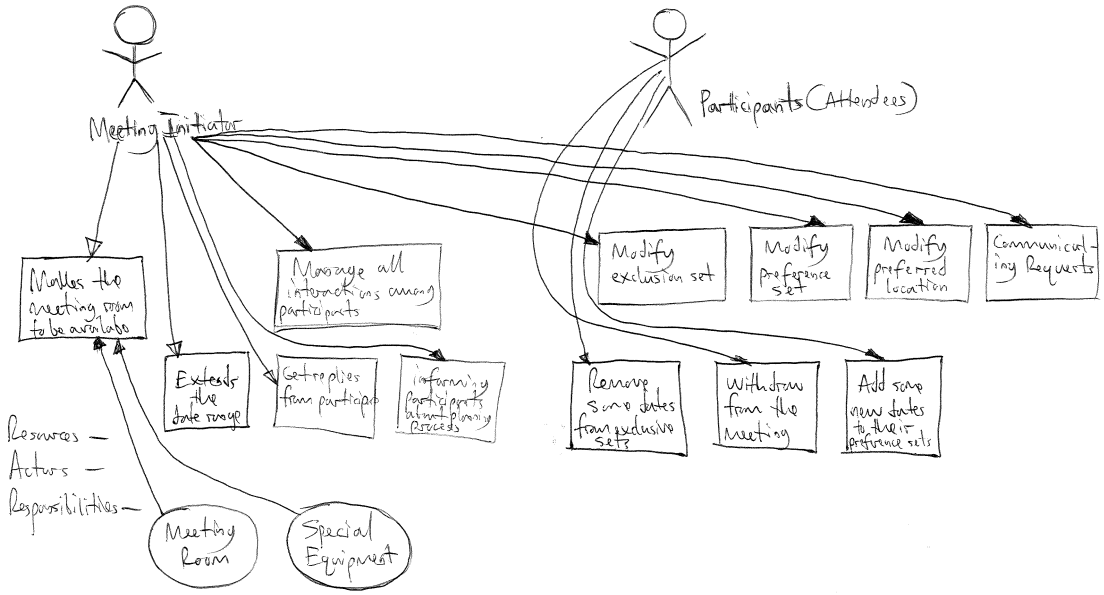


Figure 8.8: Participant F's responsibility model

notation and style presented in the tutorial. Their model used relation types not defined in the tutorial (such as resource to resource relations), while others defined in the tutorial were not used at all. Many of their relationships between entities are indirect and implicit rather than direct, leaving the details of such relationships unclear. However, their model does provide a very comprehensive description of the concepts and activities in the system definition; shortcomings arise in the description of the interactions between these concepts and activities instead.

8.4.6 Participant F

Participant F has previous experience with safety analysis techniques, such as STAMP [109] and Accimap [186]. They had applied a number of safety analysis techniques in a comparative case study in the context of health informatics.

Participant F's responsibility model of the meeting scheduler is shown in Figure 8.8. Their model is closely aligned to the notation presented in the tutorial, using all elements of the notation in a syntactically correct way. The model covers all the core system elements, clearly containing responsibilities for expressing and checking the preferences of meeting attendees. However, there is no interaction between responsibilities - responsibilities may be linked to an actor or produce resources, but no responsibilities are required by other resources, and no responsibilities require resources. For example, the responsibilities 'Add some new dates to their preference sets' and 'Modify preference set' appear to change the same set of information, but are not linked in any way in the model. Additionally, a clear

separation is drawn between the ‘holds’ and ‘required actor’ relations - these are always used as separate relationships, and are never combined to show that an actor is both required by and accountable for a responsibility.

Participant F adopted an actor-by-actor approach to determining responsibilities. They first aimed to identify the responsibilities of the meeting initiator before then considering the responsibilities of the participants. They used just one actor to represent participants - “I identified participants in a generic way, because obviously there might be diferent categories”; they similarly abstracted rooms details as the resource ‘Special Equipment’. They found deciding on the relations to be “tricky”, particularly the difference between ‘holds’ and ‘required actor’. They considered that the “major responsibilities” of an actor should be linked by ‘holds’ relations, while ‘required actor’ was used for more minor responsibilities. Their understanding of these relations was that they were exclusive, which is why they were never used together. This is a clear variation from standard responsibility modelling process; these relations are often used together, and the importance of responsibilities does not affect the choice of relation.

When questioned about the lack of production / consumption links between responsibilities operating using the same information, Participant E was clear that these responsibilities were related. With hindsight they agreed that these responsibilities should be linked via resources, but noted that they had been focused on modelling the links between responsibilities and actors when constructing their model.

Participant F had some difficulty in defining all “valid activities” and linking them to actors. They did not find the process of identifying the responsibilities difficult, but choosing the right actors to link to them was. This was a factor in their use of a single generic ‘Participants’ actor, as it reduced the complexity of relating actors and responsibilities. They also expressed some difficulty with the definition of resources - their final model only contains two resources, but they felt that they may have missed some others. They also noted that the most time consuming part of creating the model was understanding the system definition and then the process of converting that into a model - “It was understanding the problem domain; that’s how I really took my time, and how best to model it”. In particular, they noted that there was not a structured approach to take for creating a model. In contrast, actually learning the responsibility modelling concepts and notation was relatively quick - “Generally I didn’t find the activities or the responsibilities difficult - it’s just trying to be certain about the participants”.

They did not explicitly include an actor representing the scheduling system in their model, but were aware of the concept; they discussed the non-functional requirements that applied to it. They made a conscious decision not to include a specific actor, considering these responsibilities to be an overarching part of the system; the scheduling computer system was

considered part of the system context rather than an explicit part of the model - “I wasn’t sure whether to include that as an actor, or if it should be the domain area”. They described how many of the non-functional requirements could be not be applied to a specific responsibility or actor, and so they took a system-wide approach.

While they did not find themselves unable to model any parts of the system they were uncertain as to whether they had produced a complete model. They found adopting to the principles of responsibility modelling difficult, at least to begin with, and so took a step-by-step approach - “At first, how I was going to model was really difficult”. This left them unclear if they had modelled all the relevant elements so they focused on modelling the ‘active’ responsibilities, and were concerned that passive elements of the system may have been overlooked - “I’m not certain if I got everything completely; I had to first identify those directly related to the participants”.

Overall, Participant F produced a syntactically valid model providing a reasonably comprehensive representation of the system, with some details missing due to limited modelling, and others due to an inconsistent application of responsibility modelling principles. Confusion over the application of the ‘holds’ and ‘required actor’ relations led to unclear representations of the links between actors and responsibilities, while missing resources hide important interactions between responsibilities. Their confidence in the validity or completeness of their model was relatively low and they were able to identify their model’s shortcomings with hindsight, implying that they had achieved a strong understanding of the methodology once they had completed the study.

8.4.7 Participant G

Participant G has limited experience with standard modelling techniques such as ER diagrams and UML, but had not applied them outside of university coursework.

Participant G’s responsibility model of the meeting scheduler is shown in Figure 8.9. Their model accurately uses the notation described in the tutorial, clearly distinguishing between required actors and actors holding responsibilities. The model provides a clear description of the core parts of the system definition - the Initiator is responsible for producing an initial list of participants, which is then used by the Meeting System to request details for participants. These details are then used to schedule the meeting and create the booking, allowing participants to attend. It contains all the main functionality as described in the initial parts of the system definition, but does not include the non-functional requirements expressed later in the definition.

Participant G used an entity-identification strategy to begin their construction of the model. They initially identified the actors in the system, before moving on to resources and respon-

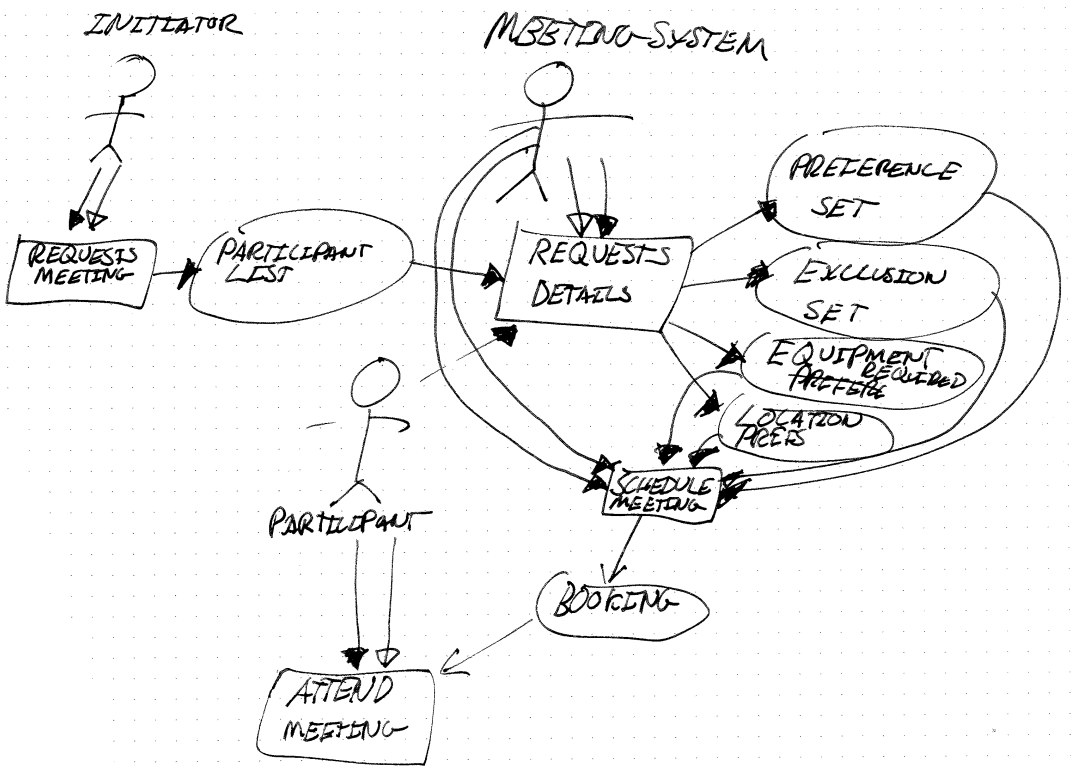


Figure 8.9: Participant G's responsibility model

sibilities and ending by determining the relations. They spent most of their time in the study actually constructing their model, and did not need to spend a significant amount of time working to understand the notation or the system definition - "the actual drawing out took the most time". They found the responsibility notation easy to understand, and did not encounter any major problems - "[Q: Were there any areas of the notation you found difficult to understand?] Not really, no".

They used an explicit actor to represent the meeting scheduling software system; they noted that software systems were described in the tutorial, and that the system definition clearly meant the scheduling system had "things to do". They had a clear understanding of the differences between 'required actor' and 'holds', expressing the reasons for each use of the 'holds' relation. Participant G had some difficulty in using resources, in particular determining the correct interactions between resources and other entities; they considered the idea of creating direct links between participants and resources, rather than having to use responsibilities.

Participant G was able to model all parts of the system that they wished to using responsibility modelling, but deliberately chose not to model some elements of the system because they did not fit with concept of responsibilities. They referred to these as "implementation details", such as the process for changing dates in response to conflicts. Such processes could be

expressed as responsibilities (such as ‘Change date to avoid conflict’) but these might not fit with a consistent level of abstraction, and excessive detail is likely to lead to a more confusing model without providing additional insight. They took a similar approach to the non-functional requirements described in the system definition - “Non-functional requirements are not responsibilities”.

They also chose to model all types of participants under the one abstract actor ‘Participant’. They had considered subtyping, but also believed that any participant that took part could be regarded as ‘Active’. They also considered having a specific actor for Important Participants, but wanted to express that this was a specialisation of ‘Participant’ rather than a completely separate actor. They were keen to see support for inheritance in responsibility modelling, favouring an approach similar to that in object-oriented programming. The version of responsibility modelling described in the tutorial materials has no support for this type of inheritance, although the concept of instantiation provides a limited ability to model this type of hierarchy.

Overall, Participant G produced a model that described the system in moderate detail and that closely followed the responsibility modelling notation that they had been presented with. They appeared to grasp the technique quickly, and did not express any significant problems in applying the notation. They chose to not to model some significant areas of the system, believing that they did not fit with the abstract concepts of responsibilities. However, the lack of simple support for inheritance in responsibility modelling prevented them from specifying the details of different types of participants.

8.5 Discussion

This section analyses participants’ models and their comments on and experiences with responsibility modelling, identifying four main areas of interest. Firstly, factors relating to the problem domain for this study are investigated; in particular the amount of effort participants required to understand the system definition. Secondly, a comprehensive analysis is performed of participants’ experience using the responsibility notation, identifying common mistakes made and shortcomings caused by limitations in the technique. Thirdly, several modelling decisions made by participants are discussed, including their process of modelling and use of abstraction. Finally, the potential influences of previous modelling experience are investigated. An overview of the more quantitative features of participants and their models is presented in Figure 8.10.

Participant	Previous Experience	Model Size	Main Effort
A	UML, ER	Large	Understanding the domain
B	Safety Cases, RE	Small	Balanced
C	Workflow modelling, UML, ER	Medium	Understanding the domain
D	UML	Small	Constructing their model
E	UML, Rational, Archimate	Large	Constructing their model
F	Safety Analysis	Medium	Balanced
G	UML, ER	Medium	Balanced

Figure 8.10: Participants' prior experience and selected characteristics of their modelling

8.5.1 System Definition

Participants generally had a good understanding of the domain as expressed in the system definition. They provided clear descriptions of the core concepts of the domain when asked, and were able to explain their models in terms of the system definition. Only the non-functional requirements (those described on the final page of the system definition) appeared to cause any problems, with some participants unclear how these more abstract details could be applied to the system.

Participants were divided over whether to include an explicit actor representing a meeting scheduler computer system. Many of the details in the system definition (such as non-functional requirements, the description of system users etc.) strongly imply that a computer system is being described, but it is never explicitly stated that the 'scheduling system' cannot be a manual process. Participants B, E and G included a system actor in their models (called 'Scheduler' or 'System'), while Participants A, C, D and F did not. Motivations for this choice varied - some participants simply considered including a system actor to be obvious, while others noted that software actors had been mentioned in the tutorial, or that the functions described in the definition were clearly those of a computer program.

Participants that did not include a specific actor often justified this by noting that their model was an abstract representation of the system, and that the nature of the scheduling system would be implementation specific. Others argued that the distinction between the system and the system context was ill-defined, and that characteristics applied to the software system could actually apply to the entire domain. Unusually, Participant D chose to not model a distinct actor responsible for scheduling, but introduced an 'Administrator'. This administrator was intended to reduce the effort required by the Meeting Initiator (the same goal as the Scheduler) but operated in a different area, assuming responsibilities that arose after the meeting had been initially scheduled.

All participants were asked what was the 'hardest' or most time-consuming part of taking part in the study and constructing their model. Most participants stated the most difficult element was understanding the system definition and the problem domain, rather than applying

responsibility modelling itself. Participant A felt the hardest part of their task was "coming up with the requirements" - the process of understanding the details of the system. Once they understood the system, converting system concepts to responsibility modelling entities was relatively easy. Participant B noted that the overly technical language used in the system definition made understanding it difficult, although they also found some parts of applying responsibility modelling difficult. Participant C described a fairly even distribution of effort - around half their time was taken to understand the definition, and the other half to construct the model. Participant F felt that understanding the technique was not time-consuming, but the combined process of understanding the system definition and deciding how to represent that using responsibilities was.

In contrast, Participant D stated that they found both responsibility modelling and the system definition easy to understand, and spent the majority of their time constructing and refining their model. Participant E also spent more time constructing their model than understanding the system definition. They produced a very detailed model containing dozens of entities, but had difficulty in attempting to express some parts of the system using responsibility modelling concepts. Participant G also found that they did not need much time to understand the specification, and were able to spend the majority of their time constructing the model.

This distribution of effort was unexpected; most of the participants already had some experience with software requirements or systems modelling, and so would already be familiar with understanding and interpreting system definitions and requirements. The systems definition was of moderate size (3 pages / 1000 words) and had been shortened from its original form [188], and so the level of difficulty attributed to it was unexpected. A difficult case study potentially makes it more difficult to determine if flaws in a model lie in the participant's understanding of a the technique or of their understanding of the domain, but a certain level of complexity is necessary to provide complex behaviour to be modelled. Difficulties with the choice of domain are not reported by Matulevičius and Heymans [122] (whose participants obtained domain information via interviews), while neither Maiden and Sutcliffe [118] or Moody et al. [127] discuss the problems encountered by their participants. Strohmaier et al. [180] did explicitly ask participants about their most difficult tasks, but used closed questions that contained only constructing or understanding models as options.

These results raise an important question as to what proportion of systems modelling difficultly and effort is attributable to the chosen technique, and how much is attributable to the choice of case study. This may affect the results of empirical studies such as this one, with the difficulty of the case study potentially being a more powerful effect than the technique being studied. Additionally, it is possible that too much focus is placed on the development of modelling techniques and notations, and that more focus should be directed at understanding the quality of requirements and methodologies for understanding them. Given these concerns, the relative impact of methodologies and cases studies on modelling is a prime area for more focused

future research.

8.5.2 Notation

Participants generally appeared to have a clear understanding of the entities in responsibility modelling, but had more difficulty understanding the various types of relations. Resources and actors were well understood, with these being common concepts in many modelling notations. Responsibilities were more difficult to understand, possibly due to a lack of familiarity. Some participants tended to avoid using the term 'responsibility' when describing their models, often referring to them as tasks instead. Several participants constructed responsibilities that do not appear to fit any definition of the term, such as Participant E's 'Domain theory' responsibility. Others modelled responsibilities that are vague or unclear (such as Participant B's 'Asks' responsibility), although they could generally describe such responsibilities in detail when asked. Participant E made the useful observation that responsibilities would require more detailed explanation as they were less familiar, and the recommendation to focus more on uncommon features when introducing a methodology is a strong one.

Participants' understanding and use of relations was more problematic, with many expressing a difficulty in understanding the concepts or applying them in a way that is not consistent with the 'normal' usage expressed in the tutorial. In particular, the distinction between the 'holds' and 'required actor' relations was difficult for some participants. Participants B and E did not use the 'holds' relation at all, and only used 'required actor' for actor-responsibility links. In Participant B's case this appeared to be due to their difficulty in understanding and applying the concepts of responsibility modelling; similarly, Participant E expressed difficulty in adopting the new concepts in this technique, including the definition and use of responsibilities. Participant F also had some difficulties applying the 'holds' relation. In their model they used both 'required actor' and 'holds', but never used them both in combination. Their understanding of responsibility modelling was that these relations cannot be used together, and so they used 'holds' to indicate an actor's most important responsibilities, and 'required actor' to indicate lesser responsibilities. However, the majority of participants were able apply the 'holds' and 'required actor' relations in a manner consistent with their definition in the tutorial.

Some participants used an incorrect or unclear notation, which made interpreting their intent difficult. Participant B used bi-directional arrows on some of their relations, and used no arrows on others. Some forms of entity links have different meanings depending on the direction (e.g. a resource can be either produced or consumed by a responsibility), and so the semantics of these relations are impossible to determine. Participant E used direct resource-resource relations; these were intended to indicate associations between similar

resources. Participant E indicated that their model also contained indirect relations, but the semantics of such a relationship is also unclear.

Participants were also asked if there were parts of the system they were unable to model due to shortcomings or missing features in responsibility modelling; some participants also offered such suggestions unprompted. Participant A wanted to be able to express choices and decisions in their model, which were not demonstrated in the tutorial. Rather than leave out this behaviour they introduced a UML-inspired decision notation to support it. For example, this was used to perform different responsibilities depending on whether or not a suitable meeting room was available. Participant B expressed a desire for a similar feature to model the implementation and process details of responsibilities. Participant E also wanted to be able to explicitly model different options in their model, which they described as 'alternative flow'. Responsibility modelling adopts a level of abstraction where it is not always necessary to specify the process taken to discharge a responsibility, but it is often beneficial to express that there are several distinct ways a responsibility can be discharged, such as when these distinct ways may consume different resources. This behaviour can be expressed in formalised responsibility modelling using the constraint logic, but this was not presented to the user study participants in order to simplify their introduction to responsibility modelling. However, this is purely a logic-based approach that does not have any graphical representation, and the notation used by Participant A appears suitable for representing the majority of constraint logic specifications. Similar notations were used in previous responsibility modelling studies for expressing the detailed workflows of responsibilities, such as in Sommerville [166].

Several participants suggested that models would be easier to understand if relations could be labelled to explain their meaning; Participant A actually included labels in their model. This is a sensible suggestion, and would be helpful for model users that have not be trained in responsibility modelling. However, care would need to be taken that labels applied to relations do not contradict the formal semantics of the relation. Labelling a 'Required Actor' relation as 'Requires' would be accordance with the semantics, but labelling it 'Performs' would introduce potential ambiguity. Alternatively, standard labels (such as 'Required', 'Produces' etc.) could be applied, which would remove the need to infer the relation type from the entities and direction of the relation.

Participant B suggested the concept of partial accountability, as they felt the current system of actors holding responsibilities was too restrictive. If an actor holds a responsibility they are accountable for it; if they do not hold it, then they have no accountability for it at all. Participant B described several scenarios where accountability was less clear-cut - for example, several actors could be accountable for a responsibility, but one of them could be considered more responsible than the others. This situation appears quite common (such as almost any responsibility in an organisation involving both junior and senior staff), and responsibility modelling currently forces full, equal and joint responsibility onto all actors

holding a responsibility. However, it is not clear how a more detailed version of holding a responsibility would be modelled, although it could be introduced into the satisfaction logic in a similar manner to ranked priorities.

Participant G strongly encouraged the incorporation of inheritance for actors into responsibility modelling. They wanted a clear way to model the various types of participant (Active, Important etc.) separately, while also indicating that all these actors were also the basic Participant. This type of inheritance is a core feature of object-oriented programming, where it allows specialisations to be developed that retain the features of a common parent. In standard responsibility modelling no such feature exists, but formalised responsibility modelling introduces the concept of instantiation, which provides a means to map between different levels of abstraction. Instantiation is best suited to cases where two separate models of the same system at different levels of abstraction are linked. This is not suitable for the Initiator example, as it is only actors that are defined at different levels.

However, there are difficulties in adopting object-oriented style inheritance more broadly. In responsibility modelling, inheritance would be semantic (subtyping), and sub-actors could replace super-actors. Some important distinctions are unclear. For example, would multiple inheritance be supported, and should inheritance apply to entity types other than actors? More generally, it is unclear what benefits inheritance hierarchies of actors provide, other than increasing the visual clarity of models. Responsibility modelling actors represent roles, and hence one real-world entity (an individual or organisation) may fulfil multiple roles; for example, someone could be both the meeting initiator and a participant. It is therefore already possible for the separate representations of participants (Active, Important etc.) to represent the same real-world actor, and the benefits of inheritance appear limited.

Formalised responsibility models that we have produced (such as those used in the case studies) tend to make extensive use of intermediation - relationships between actors and resources inevitably feature a responsibility, as the formal definition of responsibility modelling does not allow direct relations between these types. Several participants disregarded this in order to produce direct relationships between actors and resources, presumably indicating the resources are produced by the actor, while others would have liked to do so if the notation allowed.

Participant A's model featured direct links between actors and resources, such as allowing the Initiator to directly update the Meeting Schedule. Participant B used direct links between actors and resources to indicate the information used by the scheduler to arrange a meeting, but they were unclear as to the semantics of this relationship. Participant C also considered using non-intermediated relations, but for more specific reasons. They wanted to be able to distinguish between singular and plural resources. More specifically, they wanted to indicate a direct correspondence between the number of participants and the number of preference

sets. This is not currently possible in responsibility modelling, as this link would have to be expressed via a responsibility producing the preference sets. Participant D did not make use of these direct links, but did express that in some cases they would have liked to so that the complexity of the model could be reduced. Participant E's model used a number of direct links between actors and resources, again with unclear semantics. Their concept of indirect links between entities provides a good understanding of which entities interact with each other, but makes it more difficult to understand causal links.

The use of responsibilities as an intermediary is an important part of automatic analysis, which uses responsibilities to evaluate properties of the system. Direct links between actors and resources would require extensive changes to existing techniques, such as overload analysis. However, a 'syntactic sugar' approach could allow the use of direct relations when constructing and displaying a model, which would then be converted to an Actor-Responsibility-Resource structure when performing analysis.

The majority of participants did not attempt to use the RESME toolkit to construct their model. Only one participant fully used the toolkit, and two others attempted to use it but encountered installation or configuration problems. The early-development nature of the toolkit makes it relatively difficult to configure and problems on specific platforms are likely, but the majority of participants did not begin the installation process. Participants were not provided with face-to-face assistance in installing and using the toolkit as it was felt this might introduces biases in modelling style and methodology. However, the perceived complexity of installing the toolkit was underrated. Some participants described the installation instructions as complex and were discouraged by them, while others did not appear confident or comfortable with manually installing software regardless of the instructions. As a result most participants drew their model in a general-purpose editor or on paper, which led to more variation in notation than if the toolkit had been more widely used.

8.5.3 Modelling

Participants' models demonstrated a very wide spread in terms of modelling detail and levels of abstraction; one model consisted of just seven entities, while another featured more than 50 entities. The size of models does not appear to correlate to any other major factors; participants who expressed some difficulties with responsibility modelling produced both large and small models, as did participants with previous systems modelling experience. Participants clearly had different interpretations of the appropriate level of abstraction, based on a variety of factors. Some participants with simpler models did so because they wanted their model to be easy to understand, or because they felt more detail was unnecessary. Others chose not to model some parts of the system because they did not fit with the concept of responsibility, considering parts of the system to be 'implementation details' not suitable for

modelling as responsibilities. Some simple models were also due to the participant's limited time, or their lack of confidence applying the technique.

While there is substantial variation between models, some trends were clear. All but one participant did not model the non-functional requirements that the system should aim to meet. Modelling these without an explicit actor representing the system is very difficult, but the majority of participants using an explicit system actor also did not include them, modelling only the functional interactions of the system. All participants used the same basic types of actors: an initiator, participants (possibly divided by type) and possibly assisting actors of some form (such as a scheduler, representative or administrator).

Several participants raised the issue of completeness; they were either aware that their model was incomplete, or concerned that it might be. This lack of completeness was not due to the abstraction of details, but arose because participants chose to or were unable to model certain parts of the system. Some potential incompleteness was also introduced by ambiguities in the system definition or difficulties in understanding that definition - for example, Participant C noted that they had unused resources, because the definition described how the information was gathered, but not what it was used for. Participants were aware of this incompleteness but were unable to explicitly express it in their models. As a result, any user of their model could consider it a complete and accurate representation of the system, and would not be aware of the modeller's doubts as to its completeness, potentially leading to incorrect or overly confident interpretation. The explicit expression of incompleteness can be achieved in responsibility modelling using the concept of explicit uncertainty, but this was not introduced to the participants.

Many participants used short descriptions for responsibilities and resources, such as Participant D's 'Meeting' or Participant B's 'Asks'. These descriptions are short to the point of being confusing; it is not clear from the model itself what activity is being described without making reference to the system definition. However, in each of these cases the participant was able to describe in detail the one-word responsibility, making clear that they did understand that part of the system. Their reasoning for choosing such short names is unclear; most participants referred to the desire for simplicity. The tutorial given to participants used short descriptions (the longest being just nine characters), and it is possible that the short descriptions in this abstract example encouraged participants to use short descriptions in their more substantial models.

Most participants used some form of entity identification as the initial stage of their modelling process; for example, Participant C worked through the system definition, underlining and annotating each concept they identified as a responsibility, resource or actor. Entity identification was then followed by the construction of relationships between this entities, which led to an initial system model. Some participants then performed a review or refactoring of their

model, such as Participant D's simplification of their initial model by reducing the number of entities.

Some participants requested a more detailed guide to the process of constructing a responsibility model; the tutorial focused on describing the concepts and notations of responsibility modelling, rather than specifying a specific process for constructing models. There were two main reasons that such a guide was not provided - the tutorial was kept short to ease understandability, and it was felt that the optimal strategy for constructing a model was variable, depending on the case study and the preferences of the modeller. However, it is clear that participants felt that they would benefit from a clearer guide, and the misuse or absence of certain responsibility features could be addressed if they were included as specific steps in a structured modelling process. Dardenne et al. [36] provide a clear seven-step process for constructing models in KAOS, and a similar approach could be adopted for responsibility modelling.

8.5.4 Previous Experience

Participants were asked several brief questions about their previous experience with modelling notations, both specifically in the socio-technical / systems modelling field and in information modelling more generally. These questions were asked to investigate if their past experience influenced their use of responsibility modelling, such as by causing them to produce responsibility models in a particular style or by affecting their speed of uptake. Overall there are no conclusive links between previous experience and participants' application of responsibility modelling, although there are several areas of interest.

There is some evidence that modellers previous experience affects the structure of their models; for example, Participant A's model shows a layout and heavy use of decisions in a style very similar to a UML activity diagram. However, other participants' models do not show an influence from the techniques they are familiar with - Participant F's model shows no characteristics distinctive of systems theory safety models, and Participant E's model features a very complex structure that quite distinct from the enterprise modelling techniques they were familiar with. The participants that produced models closest in style to the 'classic' responsibility model structure consisted of both experienced and inexperienced system modellers, suggesting that the effects of past modelling experience are not especially strong. Participants with more modelling experience were more likely to identify shortcomings in the reduced version of responsibility modelling that they were introduced to, but their requests for extra features varied widely, without a clear link to the techniques they had already used.

Additionally, there does not appear to be a clear link between participants' past modelling

experience and their difficulty understanding or applying responsibility modelling. Participants with substantial systems modelling experience appeared more likely to report some difficulty in understanding or applying the more complex parts of the notation (such as the difference between 'holds' and 'required actor'), perhaps because these distinctions were not present in the techniques they had used before. However, this group of participants were more likely to produce large models, which suggests that any difficulties in understanding did not deter them from applying responsibility modelling. Participant E observed that they found responsibilities more difficult to understand than the other entities (actors, resources) which they had encountered in other modelling notations, but this does not suggest that they found responsibilities inherently difficult to understand. However, this does suggest that tutorials and guides for responsibility modelling should place extra focus on the concepts that are not widely used in other modelling and analysis techniques.

8.6 Threats to Validity

There are clear issues with the use of university students as experimental participants. Students are relatively homogeneous in background and demographics, and are unlikely to be representative of any wider sample. The ideal sample of participant for this type of experiment is unclear and often left undefined or only implicitly stated; in this case the ideal sample would consist of modellers and analysts from across different domains and with different backgrounds, including requirements engineering, safety analysis, organisational and business modelling etc. Our sample of students differs from this in several important ways; it is likely to be overweighted for experience with computing science modelling techniques (such as UML diagrams and entity-relationship modelling) and short of those with any background in other forms of modelling. As a result, students may deliver better results for techniques that are similar to those they have recently studied, and poorer results where a technique clashes with their previous experience. However, others argue that use of student samples does not adversely affect external validity [45].

Despite this, it is noted by Moody et al. [127] that most experimental studies of conceptual modelling use computing science students, presumably due to their availability to the researchers. No specific effort was taken to obtain a representative or random sample of participants as convenience sampling was used. Given that the study is explorative, convenience sampling may be appropriate [164].

Obtaining non-subjective assessment of participants' understanding and effective application of a modelling technique is difficult. Analysing models they produce provides a reasonable indication of their understanding and skill at usage, but is subject to potential risks. Firstly, assessing a model is itself a subjective task as differences in scope, detail and style all cause

a model to vary, and the optimal choice of these variables varies between each modeller or analyst. Secondly, a participant's model may not correlate with their level of understanding - the quality of their model might be influenced by factors unrelated to their understanding, such as the amount of time available to them. Alternatively, asking direct questions to participants (via interview or questionnaire) about their understanding relies on their own self-assessment of their knowledge, which may be unreliable. This study uses a combination of inspecting participants' models and asking indirect questions; this provides some protection against unreliability and subjectivity, but a level of subjective interpretation remains.

Some characteristics of the participants were considered as potential factors that might explain the variance in participants' models, but were ultimately discarded. The native language of participants was considered, as non-native English speakers might require more time to interpret the problem definition or have more difficulty understanding it. However, there is no clear common pattern amongst the three non-native speakers - their estimation of effort is not notably different from the native speakers. Similarly, there are no obvious model features that appear disproportionately in the models of non-native speakers.

Our choice of case study also induces potential threats to validity. The meeting scheduler example is familiar in the literature and has been modelled before, which allows comparison with other modelling techniques. The description has only been lightly modified from previous versions, which limits the risk of experimental bias in constructing a problem definition that is particularly suited to responsibility modelling. The meeting scheduler is broadly representative of the wider set of socio-technical problems - it has a reasonable amount of depth, and contains both technical, individual and organisational roles. However, it is small compared to the problems considered in Chapters 6 and 7 (although this a inherent flaw to case studies with limited resources) and lacks any particular flaws, holding a strong degree of internal consistency. As a result, the external validity is partly limited.

The sample size of this study is small, which limits the generalisability of its findings. However, we do not argue that this is a representative sample of modellers and hence draw general conclusions about the application of responsibility modelling by such modellers; instead, the study examines the experience of each participant individually to identify points of interest without inferring that they are applicable to all modellers. The purpose of this exploratory study is to identify points of interest around responsibility modelling that may not have been previously considered and to provide initial information about modellers' experience, which can be further investigated in more substantial studies.

The participants' inconsistent use of different modelling or drawing software may affect the results of this study if these packages influence the models that participants construct. For example, the RESME toolkit enforces the use of relations in the exact format specified, eliminating the possibility for syntactic errors. In contrast, participants drawing freehand on

paper can construct any notation they wish. Examples of this were apparent in the study, such as Participant A's use of UML-inspired decisions in a drawing package that provided special support for UML.

The uptake of the RESME toolkit was overestimated in the planning of this study, and this may contribute to the wide range of notations used by participants. Some interesting results were obtained by its lack of use (such as the demonstration of several alternative notations), but the use of a standard tool would be more representative of real-world usage. Participants appeared to be discouraged by the complexity of the installation process of RESME (potentially requiring the installation of Eclipse and additional dependencies), and apparent incompatibilities with specific Eclipse configurations were also problematic. This could be averted in future studies by providing participants with a computer preconfigured with the toolkit (such as in a computer lab), although this would likely require that the experiment be completed in a single block of time.

8.7 Conclusion

This user study has investigated non-expert modellers' understanding and use of responsibility modelling. Seven participants took part in an exploratory study where they were introduced to responsibility modelling and produced responsibility models of a meeting scheduling system. Participants were then interviewed to determine their experiences of using responsibility modelling, and the models they produced were analysed. Overall, this study suggests that responsibility modelling can be applied by non-expert modellers with a reasonable degree of effectiveness, although this is subject to a significant variation between participants. A number of issues around the notation and explanation of responsibility modelling, as well as the process of constructing a model were identified, and are candidates for further investigation in future work.

Two main research areas were addressed by this study. Firstly, 'Can the responsibility modelling technique be understood and applied by non-expert modellers?'. This user study has found that non-expert modellers were generally able to understand the main concepts of responsibility modelling and construct models. Participants understanding of the core concepts was good, although some features were more difficult for them to understand. In particular, the abstract nature of responsibilities was more difficult to apply for some participants, and the distinction between the 'required actor' and 'holds' notation was often unclear. All participants produced a model using the main features of the responsibility modelling notation, although there was significant variance in the scope of these models and the fidelity to the presented notation. In particular, the very limited use of dedicated tooling led to the use of a variety of inconsistent notations.

Secondly, ‘Can non-expert modellers use responsibility modelling to produce models that represent their view of a system?’. Participants were able to construct models that described the majority of the behaviour they wished to model, providing a close but not complete representation of their view of the system. Participants that could not represent their complete view of the system expressed distinct requests for extra features, including inheritance, explicit choices, enhanced accountability and labelled relations. Interestingly, a majority of requested features are already contained within the full version of formalised responsibility modelling.

Several important issues have also been identified that could be further investigated by more specific studies. Participants suggested that they spent a significant proportion of their time understanding the problem domain, rather than constructing their model. This implies that improving the usability of modelling techniques will lead to diminishing returns, as the time taken to understand the domain is not affected. The effect of the level of detail of a technique on understandability is also important to understand; some participants had difficulty understanding the basic concepts introduced to them, but others were limited by the omission of more advanced concepts that had been removed for simplicity. Finally, the general understandability of models produced by the non-expert modellers was not assessed; they were able to construct models that represented their view of the system and that we could understand, but their models might be difficult for less experienced viewers to comprehend.

Chapter 9

Conclusion

9.1 Introduction

This thesis has presented a series of improvements to responsibility modelling, a socio-technical modelling and analysis technique. Chapters 2 and 3 presented a survey of existing socio-technical modelling and analysis techniques, and developed an iterative methodology for constructing and validating socio-technical models. Chapters 4 and 5 provide a formalised definition of responsibility modelling based on the semantics of EMOF incorporating new concepts, as well as defining automated and semi-automated analysis techniques enabled by the increased formality. Chapters 6 and 7 describe two case studies using formalised responsibility modelling, comparing it to older versions of the technique and to the understanding of domain experts. Finally, Chapter 8 described an explorative study of modellers' experience of using responsibility modelling, identifying the strengths and weaknesses of the technique.

This chapter concludes the thesis by presenting the main research contributions, addressing the original research questions and presenting suggestions for future work. Section 9.2 addresses each of the research questions initially presented in Chapter 1, using the evidence presented throughout the proceeding chapters. Section 9.3 then summarises the main research contributions of the thesis. Section 9.4 discusses potential threats to validity of this research, examining each research question in turn. Section 9.5 provides a comprehensive examination of outstanding issues and future work in responsibility modelling and in socio-technical modelling generally. Finally, Section 9.6 concludes the thesis by providing an overall summary of the research.

9.2 Research Questions

The first research question 'Can responsibility modelling be formalised in a consistent manner?' is addressed in Chapter 4. Responsibility modelling is formalised from first principles, starting with definitions of entity and relation types. Entity definitions are used to construct clear definitions of entity-to-entity relations, which are modified from previous work to provide clearer semantics. These definitions are formalised in an EMOF (Essential Meta-Object Facility) [135] meta-model, which provides a standard system for defining the behaviour of domain-specific languages. This formal definition provides a consistent semantics for relationship between responsibilities and the definition of entity types.

In particular, super- and sub-responsibilities are restructured as required responsibilities, which generalises the relation to incorporate any direct dependencies between responsibilities while removing the need for composition, which is common but not universal in such relations. The relationship between actors and responsibilities is clarified by separating this into 'holds' and 'required actor' relationships, which differentiates between the operational requirement for an actor (required actor) and the accountability and control elements of being responsible for it (holds).

This restructuring continues with delegation and dependency, which are standardised by moving away from explicit statements of delegation. Delegation is no longer a direct act indicating the transfer of a responsibility; instead these actor-to-actor links are generated dynamically by calculating dependency, where an actor requires the success of other actors in order to successfully discharge all its own responsibilities. This broadens the application of dependency to wider multi-organisations domains (like Systems-of-Systems) where conventional delegation relationships cannot apply.

Previously it was difficult to use responsibility modelling to specify responsibilities that had complex forms of discharge, such as responsibilities that relied on a subset of possible resources or that could be discharged by different actors. This is addressed and formalised by the introduction of satisfaction criteria - a constraint logic that can be used to specify complex discharge requirements using standard logical operators. This logic allows the expression of either/or pre-requisites, enables multiple possible discharge options and can be used to specify primary and fall-back responsibilities. A default satisfaction criteria is defined to simplify modelling and analysis.

As with previous work, formal transformations are defined to allow common refinements of model elements without unintentionally changing the wider model. Additionally, formalised responsibility modelling introduces the ability to explicitly model uncertainty (in the domain or the modeller's understanding of it), allowing for the direct representation of unclear elements rather than forcing models to specify unjustifiable levels of precision.

This results in a formalised notation of responsibility modelling that is able to represent all forms of system behaviour that could be represented in previous versions, in addition to introducing support for useful new modelling features. The use of clearly defined semantics maintains consistency across modelling and analysis, and provides the first version of responsibility modelling to have complete and comprehensive semantics.

However, reducing responsibility modelling to a succinct core of entities and relations means that some potential areas of interest are not supported by this form of the notation. Outline notation extensions for these areas (such as contradictions in models) are described in future work.

The second research question 'Can automatic analysis techniques and tooling be developed for formalised responsibility modelling?' is primarily addressed by the development of tooling in Chapter 5, and the application of this tooling via case studies in Chapters 6 & 7. The formal definitions and elimination of ambiguity ensure consistency in responsibility models, which allows automation of analysis techniques that could previously only be performed manually. Additionally, clearer semantics enables more structured methodologies for non-automated analysis.

Firstly, simple local checks can be performed automatically. Resources without production responsibilities and responsibilities without assigned actors can be quickly identified, which can indicate potential system failures or gaps in the accuracy of the model. Actor overload analysis allows for easy detection of actors holding large numbers of responsibilities, which could lead to load-induced errors or discharge delays as multiple tasks must be addressed simultaneously.

Wider model analysis can also be automated. Criticality detection traverses the entire model to identify the consequences of individual failures on other responsibilities, and is used to locate the most critical entities - the entities whose failure would lead to the highest numbers of failed-to-discharge responsibilities. Such entities are clear candidates for deploying redundant systems or introducing fall-back systems. Similarly, the interconnected nature of socio-technical systems can lead to actors becoming dependent on other actors, which is detected by reliance analysis. This analyses the responsibilities held by actors that require other entities to be successfully discharged, and generates the set of actors that must active in order to discharge the original actor's responsibilities. This identifies clear vulnerabilities where an actor relies on others that it cannot influence control over. The use of satisfaction criteria allows the specification and validation of complex interdependencies between entities, which can be combined with selective disablement to investigate failure effects.

These techniques are supported by the development of a responsibility modelling toolkit as an extension for the Eclipse platform. This toolkit enables the graphical creation, editing and analysis of responsibility models using the techniques previously described. Models can be

represented graphically or textually, as well as a platform-independent XML representation. Analysis results are clearly presented using a mix of graphical highlights and textual descriptions. Entities can be selectively enabled and disabled to examine the wider effects of failures, and models can be divided into sections to allow visualisation and analysis of subsections.

The third research question 'Can formalised responsibility modelling be applied consistently and without specialist knowledge?' is addressed in both the Hunterston and TCAS case studies, but is most specifically addressed in Chapter 8, which describes a user study into modellers' usage of responsibility modelling.

The two case studies demonstrate the use of responsibility modelling when the modeller is an expert in the technique, but does not have specialist knowledge of the domain. We had little prior knowledge of nuclear safety, emergency planning or aviation safety prior to conducting these case studies. In each case responsibility modelling was used to produce useful and broadly accurate representations and analyses using consistent notation and concepts (augmented partially by lightweight extensions); in this sense responsibility modelling was applied consistently and without specialist (domain) knowledge.

However, this research question has more than one aspect. As well as considering whether responsibility modelling can be applied consistently across domains and without specialist domain knowledge it is also important to examine whether responsibility modelling can be applied effectively by different modellers who lack specialist knowledge of the technique. This is addressed by the user study in Chapter 8.

In this study participants were introduced to the technique of responsibility modelling, and used it to model a meeting scheduler system. These participants had made no prior use of responsibility modelling, although they had all applied some form of systems modelling in the past. Participants were interviewed as to their experiences with responsibility modelling, and the features of their modelling were analysed.

This user study discovered that non-expert users were able to understand the core concepts of responsibility modelling with enough confidence to apply them, although certain elements (such as actor-responsibility relations) proved difficult for a minority of participants. Conversely, some participants felt that the simplified form of responsibility introduced to them lacked important features.

Users felt they were able to construct models that represented the majority of the system that there were modelling; some types of behaviour (most notably decisions) could not be easily represented. There were few contradictions between participants' own verbal descriptions of the system and their models, providing some confidence that their models accurately represent their intended system. There was significant variation in models between participants, but this was primarily attributed to different system designs and different levels of abstraction rather than inconsistency in applying responsibility modelling. Overall, users were able to apply

responsibility modelling without requiring specialist knowledge, although some inconsistency did remain.

The fourth research question 'Does formalised responsibility modelling provide easier or more detailed analysis than previous responsibility modelling approaches?' is addressed by Chapter 6, which demonstrates the repeat of a case study previously performed using earlier responsibility modelling techniques by the InDeED consortium [86].

Firstly, it was demonstrated that formalised responsibility modelling was able to accurately capture a situation of identical complexity, and that the formal definitions of entities and relations did not hinder the expressiveness of the notation. Differences in structure between the new model and the InDeED model represented different approaches to modelling (such as choosing to focus on different parts of the system) rather than differences in the two versions of the technique.

The extra investigative power of responsibility modelling was apparent during the model analysis, where formal and semi-formal analysis techniques detected a number of potential vulnerabilities that were not apparent from inspection of InDeED's model. These included detection of potentially missing agencies, as well as responsibilities not assigned to any actor. This was made possible by the precise definition of relations in formalised responsibility modelling, enabling the definition of standard analytical techniques. These analysis implications were supported by several complementary sources, including observations from table-top planning exercises and revisions to the emergency plan made in later years.

Throughout that chapter, as well as in Chapter 7 the ease of use is aided by the deployment of effective tool support. This provides a straightforward interface for model creation and editing, but the most significant contribution is the automation of analysis, allowing complex analytical techniques to be applied without any specialist knowledge. These analytical techniques are described in Chapter 5, which outlines the full range of analysis and evaluation techniques that can be applied to formalised responsibility models.

The fifth research question 'Does the application of formalised responsibility modelling produce models and results that are comparable to domain-specific approaches or expert analysis?' is primarily addressed by Chapter 7, which applies responsibility modelling to a well-studied aviation system - the TCAS mid-air collision warning system.

The modelling process used in this case study is iterative, with each stage of modelling and analysis validated against an external source, which is then used to develop the next stage of modelling. This process helps to ensure that the resulting responsibility model corresponds to existing studies, by both using them as a both a validation source and a data source.

In Chapter 7 the responsibility model is validated initially against accident reports before being validated in detail via interviews with domain experts. The results of responsibility modelling broadly correspond to those identified in accident reports and those raised by

domain experts - many issues are identified by both responsibility modelling and accident analysis or expert analysis; others are identified by only one, but it is difficult to ascertain whether such issues are valid or not. However, discussions with domain experts raised no issues that were 'obviously wrong', providing a degree of confidence in the process. There is a lack of similar comprehensive studies of TCAS to provide a basis for comparison, as most studies focus exclusively on TCAS technical details. The results of these studies do not contradict those obtained by responsibility modelling, but identify additional issues that are not within the scope of this case study.

Given the evidence obtained from applying formalised responsibility modelling in case studies, responsibility modelling *can* produce models and results that are comparable to other techniques, but this is not guaranteed. It has been demonstrated that careful modelling and analysis can produce comparable results in specific fields, but it is possible that changes in modelling methodology could invalidate this, or that responsibility modelling may simply not be suited to particular problem domains.

9.3 Research Contributions

This thesis makes a number of contributions to the improvement of responsibility modelling, as well as to the wider field of socio-technical modelling and analysis. These include:

A methodology for iterative modelling and evaluation of case studies

Many socio-technical case studies struggle to demonstrate effective validation of their models, which potentially undermines confidence in the results and implications from these studies. These case studies are often the main method for demonstrating and validating socio-technical modelling techniques, so ineffective validation of case studies also undermines the usefulness of modelling techniques. Case studies often only demonstrate a technique or act as a tutorial as to their use, with limited attempts to check the accuracy or utility of the models (e.g. [99, 189, 204])

Chapter 3 presented a new iterative modelling and evaluation methodology that draws on action research and design science to provide an effective process for iteratively developing models while evaluating them against different data sources.

Formal definitions of core responsibility modelling concepts

Previous definitions of responsibility modelling varied substantially from paper to paper, with different sets of relations and entities, often holding different meanings. Chapter 4 demonstrated a consistent set of basic entities and relations that retains the full expressiveness of existing approaches, while simplifying the range required. Definitions were provided for each of these and the use of basic concepts to construct more complex models was discussed.

This formalisation greatly increases the ease of developing tool support for responsibility modelling, and enables the straightforward development of variations and extensions to the core concepts.

Extensions to existing responsibility modelling notation

The definition of core primitives allowed the extension of the notation with additional features and functionality. A satisfaction logic was defined that allows for the specification of complex relations between entities, incorporating common socio-technical concepts such as fallback modes and redundancy. Support was also added for explicitly modelling uncertainty and indicating where lack of information may limit analysis. Formally defined transformations are provided for common model changes such as the delegation of responsibilities.

Tool support for construction and analysis of responsibility models

Introducing formally defined semantics to responsibility modelling allows for substantial automated analysis, but this requires proper tool support. A software toolkit was developed that supports graphical creation and editing of responsibility models, and provides automated analysis on these models. The effectiveness of these automated techniques was then demonstrated through case studies.

A comparative case study of formalised and unformalised responsibility modelling

A direct comparison of formalised responsibility modelling and a previous non-formalised version was made in Chapter 6 by re-examining a case study of the Hunterston Nuclear Power Stations Off-Site Emergency Plan which had previously been studied by the InDeED consortium. Analysis using formalised responsibility modelling identifies the same core issues as the previous study, but additionally detects several potential vulnerabilities that were not previously detected.

Application and evaluation of responsibility modelling in the aviation domain

A case study was performed of responsibility modelling in aviation by modelling, analysing and evaluating the TCAS aircraft collision warning system using responsibility modelling and evaluating these results with domain experts. This provides two main contributions; firstly, demonstrating the use of responsibility modelling in a new domain; secondly, evaluating the correctness, understandability and relevance of responsibility modelling to experts in a particular domain.

Qualitative evaluation of responsibility usage by non-expert users

We conducted a user study asking participants to construct a responsibility model of a simple socio-technical system, and analyse the resulting models and evaluate participants' use and understanding of responsibility modelling. This study identifies responsibility modelling concepts found difficult by users and examines common characteristics in the application of re-

sponsibility modelling. Additionally, users' recommendations for extensions to responsibility modelling are presented and analysed.

This is (to the best of our knowledge) the first evaluation of responsibility modelling performed by non-expert users.

9.4 Threats to Validity

This thesis has addressed the stated research questions and made a number of valuable research contributions, but it is important to address potential threats to validity that are present in the research. Certain characteristics of the research are common to socio-technical studies (such as reliance on case studies), while others are specific to particular aspects of this thesis.

The semantics of responsibility modelling described in Chapter 4 are defined using an EMOF meta-model which provides formal typing and a reference implementation for modelling, making the consistency of the notation formally provable. Others elements of the notation are described in a structured format but without using an underpinning formalism, potentially allowing usage or implementations to deviate from the intended semantics. This could be addressed by providing a formal definition for these features based on an existing language such as OCL [191], although this could require considerable effort to relate the abstract responsibility concepts into operationalisable statements.

The RESME toolkit described in Chapter 5 can be easily installed and tested, and similarly the use of automatic and semi-automatic analysis methods can be demonstrated. However, it is more difficult to prove the efficacy of these techniques in real-world applications. The case studies in Chapters 6 and 7 demonstrate and validate the use of the techniques in two specific examples, but these may not necessarily generalise. Demonstrating the broad efficacy of a technique is a common difficulty in socio-technical studies, and it is common to continuously apply a new technique to different domains over a long period (such as the application of i* to different case studies over a decade [68, 112, 202, 204]).

Given its small scale, the study in Chapter 8 is vulnerable if overgeneralised. This study uses a small sample of university students, which places strong limits on generalisability. The relevance of such samples to the general population is much argued [45], but is generally considered acceptable for exploratory studies [164]. Small samples of university students have been used for similar requirements engineering studies such as [122] and [180]. The choice of case study may also affect the validity of the experiment, but our case study has been used previously by van Lamsweerde et al. [189].

The use of semi-structured interviews introduces the potential for inconsistent coverage of issues in different interviews; the use of prompts provides some protection against this, but

cannot cover all possible issues. This is mainly unavoidable in an explorative study such as this one. The main topics of interest cannot necessarily be predicted in advance, and there is a lack of similar studies to base questions on. Given the small sample size and wide range of topics in this study we did not use a formal interview coding system to structure and classify the results, instead using a narrative approach for each participant and a synthesis of important overall results. More structure and rigour could be provided by a follow-up study using more specific questions and a larger sample size.

In the study of the Hunterston Nuclear Power Stations Off-Site Emergency Plan in Chapter 6 specific steps were taken to ensure the validity of the conclusions drawn. The responsibility model was constructed solely by reference to the Emergency Plan and without any specialist knowledge of the nuclear industry in order to remove any biasing caused by outside knowledge. Materials from the InDeED project were used to compare our model of the system with that of InDeED and to act as verification of its correctness, but these materials were not read or used until after our own model had been produced. While the comparison between the two models was not carried out by an independent researcher, the process of comparison and verification against observations was performed using only open data and is easily reproducible.

Chapter 7 demonstrates the application of responsibility to aviation safety, and provides evidence that the analysis results of responsibility modelling can be comparable to other techniques applied in this domain. However, the strength of this claim should not be overestimated. In the TCAS case study responsibility modelling demonstrates a correspondence to other techniques, but this is only one application in a specific domain. The Hunterston case study in Chapter 6 also provides some evidence that formalised responsibility modelling is comparable to other techniques, as some issues identified by responsibility modelling were corrected during routine revisions of the emergency plan. However, examining this type of comparison was the not the main purpose of that case study, and so extensive conclusions should be not be drawn.

The validation of analysis results by domain experts provides strong evidence that results of responsibility modelling are valid, while a comparison with other techniques indicates that responsibility modelling is at least as effective in identifying potential vulnerabilities. As with other comparisons and evaluations in this thesis this was not carried out independently, but full transcripts for the interviews have been deposited [160]. Given the unusual structure (based around interaction and observation of a model) and content (lengthy, in-depth discussion with only a few participants) of these interviews they were not formally coded, and presented using a narrative approach.

This thesis relies throughout on the use of case studies either for demonstration and validation; three of the five research questions are addressed primarily or partially by using case studies. This approach potentially opens threats of validity if cases studies are not representative or do

not cover important aspects of socio-technical systems. The use of different case studies only provides partial protection if the method as a whole is lacking. However, reliance on case studies is common to almost all analyses adopting a socio-technical approach, and can be seen in requirements engineering [107, 203, 189], safety analysis [109], and organisational modelling [99, 148]. These case studies cover a range of domains, but the applicability of some case studies to the technique they are used to model is unclear, such as the use of the simplified mine-pump case study for a requirements engineering technique [107]. The iterative research strategy described in Chapter 3 aims to minimise some of these effects, such as allowing information sources to be used for both validation and further modelling, mimicking the use of triangulation in design science [134]. Additionally, all of our case studies have been at least partially studied by other researchers in the past, and two of the three use only source materials used in these previous studies, potentially reducing the subjectivity of our choice of case studies and increasing their reproducibility.

9.5 Future Work

The formalisations of responsibility modelling addressed many of the shortcomings and inconsistencies in the technique, but many areas of future work remain and are discussed in this section. Generally, potential changes to responsibility modelling face a difficult balance between specifying additional detail enabling improved analysis while maintaining an abstract, easily understandable and usable modelling notation.

Several potential extensions to responsibility modelling are outlined, providing additional model detail in certain areas, but possibly at the risk of increased complexity and modelling effort. Contradictions are rarely explicitly handled in socio-technical models, but contradictory requirements can be a major source of implementation problems. Responsibility modelling could allow the explicit expression of contradictory elements, but detection of such contradictions can be difficult and time-consuming. A detailed system of constraint language quantifiers is suggested, which would allow the consequences of successful but non-optimal responsibility discharge to specified and propagated throughout the model. The relative lack of use of the constraint language within this thesis's case studies is also discussed. Additionally, a more elaborate treatment of timing is discussed, which would allow the timeliness of responsibilities to be expressed in orders of magnitude.

Options for increased abstraction are also discussed. The concept of realisations from previous work by Lock et al. [116] is extended, which would allow abstract entities to be assumed by other entities. This potentially allows for useful mappings between high- and low-level models (such as between requirements and implementations), but many elements of the semantics are unclear. More generally, the option of using improved tool support to allow

multi-level responsibility models is examined, which could enable the use of submodels to provide specific details, as well as other improvements to the existing toolkit. Similarly, improved integration with other techniques could allow the use of more formal methods for the specification of particularly complex responsibilities.

Experience gained by performing case studies also showed that some aspects of responsibility modelling were not especially effective and are in need of change. Overload analysis proved to be of little utility when analysing technical actors, such in the TCAS case study. Technical systems have quite distinct functionality and performance when compared to individuals or organisations, which overload analysis does not take into account. Finding a suitably abstract system that also produces useful analysis results appears difficult, but several possibilities are investigated, including a complete reworking of the concept of overload analysis.

9.5.1 Contradictions

Previous studies of responsibility modelling have not considered the possibility that models may not be internally consistent. Models may abstract over or only partially represent the domain, but the elements of the domain that are represented are considered to be consistent (This is in contrast to accuracy, where it is acknowledged that a model is unlikely to fully match real-world operations). As a result, we assume that there are no complex links between entities other than those explicitly modelled with relations. In particular, we assume that successfully discharging responsibilities is always the correct intent of the system, and that discharging one responsibility does not negatively affect another.

However, this is not true in many socio-technical systems [165]. Responsibilities (in particular) can be contradictory, where it is either not possible or not desirable for two or more responsibilities to both be discharged. This can arise where different sources have different representations of the domain (for example, differences between two documents, or a difference of opinion between two stakeholders), or where the fundamental definition of the system is itself flawed. For example, consider an example from the TCAS case study - one chapter of the ICAO manual indicates that pilots should immediately inform the air traffic controller when they respond to an alert; another chapter of the same manual indicates that they should inform the air traffic controller only after completing their response to the alert. These two responsibilities cannot correctly co-exist, and the inconsistency reveals a potentially severe vulnerability in the system.

Ideally, any modelling and analysis technique should be able to detect these inconsistencies, and highlight them for further investigation. However, to do so with a general-purpose modelling technique is extremely difficult. Responsibilities are defined only by their relations with other entities, and their free-text name / description. This does not provide enough contextual

information for an automatic analysis to detect inconsistencies without also maintaining a detailed operational model of the domain, which would defy the logic of using a general-purpose technique. Instead, these types of inconsistencies may well be detected by analysts inspecting the model, or identified as the model is being constructed from source materials.

Extensions to the existing notation may be able to more effectively detect such inconsistencies, drawing on broader theoretical techniques. For example, potentially conflicting obligations could be analysed using a deontic logic [57], where responsibilities are converted into formal rules of action. A theorem prover can then be used to determine if the formalised representation is consistent, or used to detect other potential vulnerabilities. However, this relies on the analyst making a correct choice of which parts of the model to convert to logic, as representing the whole responsibility model in deontic logic could be time-consuming and unnecessary.

Alternatively, mutually exclusive responsibilities (probably the most common type of inconsistency) could be explicitly modelled with an additional relationship, indicating that they should not both be discharged. However, the deeper semantic meaning of such a relationship is unclear, and would require modifications to existing analytical techniques such as dependency analysis. In either case, such modifications would allow modellers to explicitly indicate inconsistencies that have already been detected, rather than aiding them in finding undetected inconsistencies.

9.5.2 Enhanced Qualifiers

Previous versions of responsibility modelling have made use of a HAZOPS-like technique using guidewords for analysing relations and entities (e.g. Lock et al. [115], Lock and Sommerville [114], Sommerville et al. [168]). These guidewords are used to assess potential system vulnerabilities by considering possible failures or degraded operation of model elements - for example, analysing the implications of the late completion of a responsibility. Guidewords provided one of the main systematic approaches to analysis in early responsibility modelling, although the results of HAZOP-type analysis can be highly subjective, especially when applying them outside of the original domain of industrial processes.

To a large extent this type of analysis has been supplanted in formalised responsibility modelling with more advanced techniques. The effects of entity failures on the wider system can be more accurately determined by satisfaction analysis, and changes to the temporal model have simplified the effects of early and late completion. HAZOPs analysis can augment these more formal techniques by analysing areas that they do not cover - in particular, they can provide helpful non-formal analysis for qualitative properties, such as the quality of resources being produced.

However, the manual nature of this technique makes it impractical for considering the indirect

effects caused by impaired performance in one part of the model - it is time consuming to consider each knock-on effect, and at each step the analyst must consider whether the earlier impairment leads to further impaired performance in the next entity considered. Despite this, the use of qualitative rather than quantitative descriptions of performance and availability fits better with the abstraction of responsibility modelling, and avoids the need to specify implementation choices in detail.

A potential solution to this problem is to introduce qualifying descriptors for entities (akin to guidewords, such as poor quality for resources and late discharge for responsibilities) into the constraint language, which would allow the effects of impaired performance to be explicitly defined in the definitions of entities. Guideword analysis could then be formalised by allowing the analyst to set the status of the relevant entities, and then automatically calculate the propagated effects of that impairment.

For example, these constraint language definitions show that poor quality steel causes the Produce Hull responsibility to be discharged in an inferior way, which then causes the resulting ship to be structurally weak:

Produce Hull:: Shipbuilder Active & Steel Exists [Poor Quality Steel -> Inferior Produce Hull]

Ship:: ProduceHull Discharged [Inferior Produce Hull -> Ship Structurally Weak]

The square brackets indicate that qualifiers are attached to that particular definition; the presence of the left-hand side qualifier in a related entity causes the right-hand side qualifier to apply to the current entity. A particular degraded state may only occur if a particular combination of qualifiers occurs; for example, a responsibility might only be impaired if both resources that it requires are late:

Authorise Expenses:: Receipt Exists & PermissionNote Exists [Late Receipt & Late PermissionNote -> Late Authorise Expenses]

However, this notation is not able to effective capture all reasonable scenarios. For example, a message could be transmitted by either email or post, or by both. Receiving the message would only be late if only one of the resources was available, and that resource was late. In this case the effects of the qualifiers are dependent on the particular state of the other constraints, and we lack an intuitive notation for expressing that dependency.

Subject to that limitation, qualifiers could be used to perform a tool-assisted analysis of the implications of impaired operation. If deemed necessary the modeller would augment an existing model with details about the propagation effects specified in constraint language. Once defined, the analyst could the state of specific entities that they wish to examine, and the knock-on effects on all other entities in the model could be calculated automatically from the constraint definitions. This approach requires additional modelling effort (to define the

constraints), but once performed it becomes possible to consider the effects of multiple sets of impairments in greatly reduced time.

9.5.3 Realisations

The transfer of obligations within responsibility modelling is well supported, normally by means of delegation of responsibilities. However, one important concept that is missing is the idea of realisation - the concept that an abstract entity defined within a model can be implemented by a more concrete entity within the same model. For example, a model might define the actor role of 'Police', and this actor could be realised by an actor representing a particular police force. In relation to actors this type of relationship was mentioned briefly in Lock et al. [116] as the 'Acts As' relationship, but unfortunately no definition or example usage was provided. The principle is not restricted to actors, and could also be applied to resources and responsibilities.

The exact semantics of such a relation are not entirely clear. The intent is to show that one entity (partially or completely) acts like and contains the functionality of another, more abstract entity. This is straightforward if we are only interested in the structure of a responsibility model, but unclear if we also wish to perform analysis. For example, does realising a resource mean that determining whether that resource is produced requires checking responsibilities that produce the original resource, or responsibilities that produce the abstract resource? More generally, does the realising entity replace the original, supplement it or not directly affect it at all? If two or more concrete entities realise an abstract one, which one should be considered for analysis?

This overlaps significantly with role analysis and actor instantiation, as introduced in Sections 4.9 and 5.6. Role analysis is effectively a simpler, less formal version of realisation that is applied only to actors. Actor instantiation extends this by providing a structured representation for mapping between actors at different abstraction levels; however, this mapping is effectively one-directional, as adding an instance actor has no effect on the original actor. This avoids the problem of unclear semantics, as role analysis is informal and manually performed and actor instantiation only visualises, rather than analyses. However, substantially greater automation and analytical power could be achieved by a formal relation that connects all types of entities, if a solution to the inconsistent semantics can be found.

In particular, this concept offers great potential when applied to requirements specification and system design. It is often difficult to make a structured argument that a proposed system design provides a full implementation of the requirements, as the languages and notations used are likely to vary significantly between the statement of the requirements and the description of the proposed system. If both the requirements and proposed system design are specified

using responsibility modelling, realisation relationships can be used to map between the design and the requirements - if all requirements are mapped then it is possible to prove that the design does meet the requirements, subject to the level of detail in the model.

9.5.4 Enhanced Temporal Behaviour

Responsibility modelling has previously used an informal approach to timing, where the time taken to discharge a responsibility has not been the subject of modelling or formal analysis. Chapter 4 introduced the concept of 'epoch time', which provides a clearer definition of this abstract form of timing.

This provides a useful abstraction over time, which is often left imprecisely specified in socio-technical modelling. In some contexts modellers may wish to specify timing in increased detail, without resorting to the use of a fully timed logic. In these cases, an approach based on the concept of Timebands [24] can be used, which provides a formalisation for explicitly modelling different levels of time. Timebands represent orders of magnitude in time. It is not necessary to define exact durations for events, but modellers can specify whether discharging a responsibility takes a matter of seconds or if it takes several hours.

If a responsibility model is annotated with timebands for responsibilities it becomes possible to determine if responsibilities will fail due to relying on other responsibilities that cannot be discharged in time. The model is analysed in order of increasing timebands: first responsibilities discharged in the shortest timeband, then the second-shortest and so on. If a responsibility at a lower timeband depends (without any alternatives) on a responsibility, actor or resource that is only available at a higher timeband then a system vulnerability can be detected. Within a timeband the traditional concept of 'eventual discharge' remains; the set of elements at each timeband can essentially be treated as a separate but interacting system, and the whole system as a composite system-of-systems.

9.5.5 Application of Enhanced Notation

The extension of responsibility modelling to allow the specification of complex behaviour using a constraint language enables the modelling of certain types of system behaviour that were not supported by previous versions of responsibility modelling. However, relatively little use was actually made of these concepts in the case studies presented in this thesis.

The lack of explicit uncertainty can be partially attributed to the domains chosen for the case studies - both were well defined systems that were modelled using formal documents as the primary information source. In these cases it is to be expected that uncertainty should be minimal, as clear uncertainties would already have been detected and corrected. As a result,

explicit uncertainty was only applied when there was clearly complex behaviour that could not be clearly defined (such as the pilots' duty to ensure safe flight). If models are constructed from less formal sources (such as observations of an existing system) then more behaviour will be apparently unexplained, and hence more explicit uncertainty is likely to be modelled. However, the choice of when to specific uncertainty is still subject to some subjectivity by the modeller. In effect, applying explicit uncertainty indicates that the modeller believes that there is sufficient uncertainty in a responsibility or relationship to affect the discharge of the relevant entity; this is partially subjective, as different modellers may have different implicit thresholds for what counts as 'sufficient'.

This explanation is less convincing for the low usage of other constraint language features, such as OR relations and priorities. In some parts of these systems the source documents indicate very clear assignments of one actor only to a responsibility, but this is not universal. There are clear examples in the chosen domains where these concepts could be applied - for example, we could state that flying an aircraft should primarily be discharged by the Pilot Flying, but that the Pilot Not Flying should step in if the Pilot Flying cannot discharge that responsibility. Similarly, we could state that either the Pilot Flying or the Pilot Not Flying could respond to ATC messages. Nonetheless, this approach was not adopted in practice. This can be partially attributed to the strict separation described in the ICAO manuals - each participant has their own distinct roles, with little flexibility. However, the lack of use can also be attributed to a choice of modelling style.

Different styles of modelling can be used to represent the same underlying elements, which leads to variability in models. A scenario where two actors can independently perform the same task could be modelled as one responsibility, with the independent nature modelled as an OR relationship. Alternatively, it could be modelled with two separate responsibilities, each held by a different actor and requiring no use of the constraint language. These are functionally equivalent in most ways (consider a responsibility that produces a resource, for example) and both are valid responsibility models. Each approach has strengths and weaknesses - the use of multiple entities is generally simpler and easier to understand for small sets, but quickly becomes intractable as relations get more complex; the constraint language is more effective for large sets of entities such as a voting circuit. The tradeoffs between these different styles of modelling (and the overall effectiveness of the constraint language) are a prime candidate for future work.

9.5.6 Tool Support

The ability of tool support to provide different visualisations and abstractions of responsibility models aids ease of modelling and understandability, but the RESME toolkit's support for this is limited by certain technical restrictions of the underlying frameworks. Section diagrams

were introduced to allow models to be easily broken down into separate sections, but are restricted as entities can only be part of at most one section. This is a limitation of the EMF framework used to store models - EMF uses a child/parent structure, and it is impractical to assign multiple parents to the same entity. This restriction causes problems when the boundaries of sections are not clear, or where there is significant interaction between different areas of a model - both of which are very common in socio-technical modelling. It makes modelling exchanges of resources difficult, and prevents overlapping views in the same model. This is purely an implementation problem that can be addressed by changes to the internal representations.

Alternatively, a broader solution would be to allow responsibility models of different parts of a system to be modelled completely independently, and for the different parts to be linked only when necessary, such as for system-wide analysis. In particular, this approach offers potentially good support for systems-of-systems problems, where each part is itself a full and distinct system. The challenge of this approach is correctly linking the separate models when required, which requires identifying entities that appear in different models. Composing models across different levels of abstraction is a difficult problem, and has been identified as such in different techniques such as modular safety cases [98]. This could be partly automated (such as matching entities that have the same name) but any practical implementation of this approach requires a user interface and underlying mapping system for defining these equivalences.

Similarly, there are very clear advantages to allowing responsibility models to contain different levels of abstraction and for these levels to be explicitly linked. This offers additional expressive power in modelling by making it clear that some parts of the model are more abstract than others, and would allow for the expression of realisation relationships, assuming that the a consistent semantics can be developed. In particular, this can be used to map between designs and implementations, or between rules and guidelines and operational procedures. Automatic analysis techniques can be also be updated to produce more accurate results by taking into account different levels of detail - for example, criticality analysis could apply higher weights to responsibilities that are realised as multiple, low-level tasks. Implementation of this form of abstraction in the current toolkit is in principle straightforward, but faces serious practical obstacles. There is a difficulty in assigning entities to a discrete abstraction level (somewhat similar to the problem with sections) as EMF lacks effective constructs for hierarchical layers; this is not a problem when expressing that one entity is an abstraction of another, but makes it extremely difficult to identify all entities at the same level of abstraction, which makes updating the automatic analysis infeasible.

Support for a more elaborate model of abstraction would also enable implementation of superior role analysis through realisations. Concrete real-world individuals, organisations and devices could then be mapped directly to responsibility model actors, which enables overload

detection to be expanded from actors to their corresponding real-world realisation; this allows the detection of actors that are only overloaded due to fulfilling multiple roles. Additionally, such a mapping would provide much clearer indication of potential conflicts of interest where one agent takes multiple, linked actor roles.

Another area where the toolkit is currently lacking is the ability to clearly visually represent the more advanced features of the constraint language. For example, the toolkit currently has no ability to display the order of priorities, or to graphically show how relations are grouped into prioritised options. These visualisations are not difficult to conceptualise (for example, groupings could be indicated by coloured highlighting when an entity is selected) but face implementation difficulties due to the restrictions of the Sirius framework.

Finally, the user studies conducted in Chapter 8 identified that some participants had difficulties installing and using the toolkit. Wider application of responsibility modelling and the RESME toolkit would be enhanced by simplifying the installation process, improving the understandability of the tool's documentation and increasing the robustness of the toolkit with regards to different system configuration and Eclipse versions.

9.5.7 Integration with other techniques

The range of applicability of responsibility modelling is one of its greatest strengths; responsibility models can incorporate both specific details and broad social duties. However, there are clearly times when responsibility modelling is not the most appropriate technique for analysing a problem; more robust analysis of low-level details may be obtained by formal logics and calculi, while it may be more natural to represent organisational details using business process or organisational models. In particular, the lack of common low-level modelling features such as timed logic and resource states limits the ability of responsibility modelling to fully capture the complex details of many processes.

This could be addressed by developing guidelines and methodologies for combining responsibility models with other techniques as part of a structured modelling and analysis process. For example, responsibilities that occur at the same time and may interfere with each other's completion could be separately modelled using a process calculi such as CSP (Communicating Sequential Processes) [78] or Timebands [13]. The results of responsibility modelling analysis could then be used to set the initial variables in the CSP script, and the results of analysing the CSP model (using a model checker) used to determine whether or not the responsibilities were successfully discharged. Similar mapping and combination projects have been developed for other techniques, such as mapping between BMPN and KAOS models [101]. Responsibility modelling integration with low-level techniques offers more advantages (expressive power, model checking etc.) than integration with higher level techniques, due to the differences in abstraction.

9.5.8 Overload Analysis

In Chapter 5 overload is defined as occurring when an actor is required for more than a certain number or load of responsibilities, where this threshold is arbitrarily set. This provides a simple mechanism to identify the actors that are under the greatest theoretical load, and are therefore likely to be at risk of impaired performance due to overload. This approach is similar to that taken by Sommerville [166], and has the advantage of relative intuitiveness. However, this does not take account of the capabilities or capacities of each actor, so every actor is assumed to have the same ability to handle multiple responsibilities. This can limit the usefulness of overload analysis, by generating warnings that are inconsistent in their severity - a large organisation required for five responsibilities could generate a warning, while a single individual required for four of those five responsibilities would not trigger a warning, even though they are proportionally much more likely to be overloaded.

Additionally, this definition of overload does not produce reliable warnings for technical actors. Mechanical, electronic and computer systems do not experience overload in the same way that humans do - they are normally designed to provide specific functionality and to operate at a certain performance level. Therefore they do not run the risk of reduced performance due to the number of responsibilities they must discharge, as long their responsibilities remain within their specification. However, in contrast to human actors their ability to handle tasks that they were not designed for is much reduced, and so it possible for such an actor to fail to discharge even a small amount of responsibilities if it lacks the capabilities required. Additionally, a technical actor holding multiple disparate responsibilities is likely to be a particularly complex system, which may make it more error-prone in design or implementation.

Attempting to address these deficiencies in overload analysis is difficult. The differing capabilities between actors could be addressed by assigning each actor a 'load-bearing' score, indicating the amount of responsibility load it can cope with. This would allow the different capabilities of more and less resourceful actors to be distinguished, but requires additional modelling effort and further increases the problem of accurately estimating the load of responsibilities and the ability of actors to cope - adding more layers of detail reduces the abstraction of responsibility modelling, and requires precise information that may not be available. Additionally, this approach does nothing to address the issues around technical actors, where load is based on specific capacities and functions rather than on a general level of effort.

Modelling the individual capabilities of each actor and the requirements of each responsibility is the most comprehensive solution to this problem, but requires an impractical amount of modelling effort and violates the responsibility modelling principle of abstraction. The potentially unique nature of capabilities means that a comprehensive capability model would require considering the relationship between each actor and each responsibility individually -

which would completely eliminate the efficiency advantages provided by automated analysis.

Alternatively, load analysis could be redesigned to focus not on the absolute values of load (the number or significance of responsibilities held by an actor) but on the relative distribution of responsibilities between actors. Under such an analysis large differences in load between actors would be highlighted, such one actor being required for just one responsibility when another is needed for half a dozen. This would not highlight whether individual actors are overloaded or not, but would instead highlight whether or not responsibilities within the system are distributed equally. A system that places many responsibilities upon a small subset of actors while the others have relatively little load indicates that effort within the system is not being distributed in a balanced way, and that there might potentially be spare capacity that could be used to share the load more evenly. An absence of distribution warnings would indicate that the system is relatively uniform in the distribution of responsibilities; problems of overload might still occur, but not because the load was distributed unevenly. This approach would allow the consideration of the system-wide distribution of load, but would not replace conventional overload analysis's checks on individual responsibilities.

9.5.9 User Studies

The user study in Chapter 8 identified several unexpected results that are worthy of investigation through further user studies. Additionally, the results of this explorative study suggest some areas of improvement to responsibility modelling in general.

The study suggested that users spent a substantial proportion of their time (for several participants, the majority) understanding the problem domain as expressed via the system definition document, rather than in the process of understanding or applying responsibility modelling. This result is counterintuitive; the participants had no prior knowledge of responsibility modelling, but had all applied at least basic modelling techniques before. This may be attributable to the structure of the study (participants were guided through the responsibility modelling tutorial, but unaided with the systems definition), but it is still surprising that some participants took more time to understand the domain than to construct their model. To the best of our knowledge this finding has not been reported in the literature of socio-technical modelling. If this distribution of effort is typical then there are important implications for systems modelling; for example, should less research effort be focused on the use of modelling notations, and more on the techniques, mechanisms and strategies used by modellers to understand scenarios? However, it should be noted that this study used only self-reported effort to draw this conclusion, and a specific study measuring participants' distribution of effort could obtain much stronger results.

The study also suggested that finding the correct balance between complexity and understandability in a technique, or the explanation of a technique, is difficult. Some participants

requested advanced features of formalised responsibility modelling that they had not been shown, while others had difficulty understanding and applying the limited set of concepts that they had been shown. In particular, there was no obvious correlation between previous experience and the desire for advanced features, which makes choosing an appropriate level of detail for presenting responsibility modelling difficult. Some participants suggested that an introduction focusing on concepts unique to responsibility modelling would be more effective. A focused study exploring modellers' understanding of responsibility modelling and identifying the most important features could improve responsibility modelling's uptake and usability.

Additionally, one important part of the responsibility modelling ecosystem remains uninvestigated. This thesis has formalised the notation, provided automatic analysis techniques and assessed the understandability of expert-produced models by domain experts. This user study found that non-expert modellers could produce responsibility models broadly matching their view of the system, but it is not clear if these models can be understood by domain experts or by model users in general. This could be investigated by a further user study, potentially using an experimental structure where models are swapped between participants.

9.5.10 Industry Uptake

A common problem with socio-technical modelling techniques is that they are developed within academia and rarely applied outside of it. This is true of 'classic' requirements engineering techniques such as i* and KAOS, and also applies to responsibility modelling; there are no known applications of responsibility modelling outside of university projects. This lack of integration between academia and industry is not unique to the socio-technical field, but several distinct issues occur.

Much research on methods is focused on the development of new techniques and extensions to existing methods; research on the effective application of existing techniques is less common, and the application of methods in real-world settings less so. This often occurs for understandable reasons (such as commercial confidentiality limiting how much activity with industry can be published), but leads to academic and industrial practice operating almost completely in parallel, without interaction. Additionally, socio-technical modelling is more dependent than many other techniques on effective tool support, but most tool support developed in academia act as prototypes rather than as finished, refined products. This is again perfectly understandable in context (the incentives to develop high-quality, production-ready software are very limited), but acts as another barrier to external uptake.

Ultimately, increased industrial uptake of a technique such as responsibility modelling is dependent on multiple factors, some of which are outside of the control of its proponents.

However, a number of steps can be taken that aim to increase the wider uptake of responsibility modelling. Improvements to the existing tooling to reduce barriers to entry (or potentially the production of an entirely new toolkit, such as web interface) would enable interested users to examine and test the technique much more easily. The case study in Chapter 7 was conducted in partnership with a industrial consultancy, although their primary role was to provide feedback on the utility of the models and modelling technique, rather than to directly apply responsibility modelling themselves. However, further small-scale engagement with industrial partners may provide an entry point to introducing responsibility modelling to wider industrial audience.

9.6 Conclusion

This thesis has presented a comprehensive body of research on the formalisation, extension, validation and application of responsibility modelling. The formalisation of responsibility modelling enabled the extension of the notation and the introduction of automated analysis, and the effectiveness of such techniques was demonstrated in real-world case studies. Additionally, both the understanding of responsibility modelling by domain experts and the application of responsibility modelling by non-expert modellers was examined.

This thesis shows that responsibility modelling can offer a useful compromise between complex or domain-specific low-level techniques and abstract, unprovable high level modelling notations. Formalised responsibility modelling aims to increase the usefulness of responsibility modelling by increasing the power of the technique without reducing its ease-of-use; formalisation and automatic analysis provide consistency and greatly improved analytical power without any substantial changes to the core concepts of the technique. This increases the range of scenarios where responsibility modelling can be effectively applied, without reducing the understandability and intuitiveness that distinguish it from other systems modelling techniques.

However, systems modelling still requires a careful consideration of the trade-off between abstraction and detail. Understandability, flexibility and breadth of application all encourage the use of abstract models and modelling techniques, but automation, accuracy and analytical power are more easily obtained by using more complex techniques and constructing more detailed models. The balance between these two approaches is dependent on the system to be modelled, the intended outcomes of the modelling and the skills and attitudes of the modeller. By demonstrating both the analytical power and the flexibility of responsibility modelling we hope to encourage the development and use of such techniques.

Bibliography

[1] Nicholas Abercrombie, Stephen Hill, and Bryan S Turner. *Dictionary of Sociology*. Penguin, third edition, 1994.

[2] Marilyn Jager Adams, Yvette J Tenney, and Richard W Pew. Situation Awareness and the Cognitive Management of Complex Systems. *Human Factors*, 37(1):85–104, 1995.

[3] Rachel F Adler and Raquel Benbunan-Fich. Juggling on a high wire: Multitasking effects on performance. *International Journal of Human Computer Studies*, 70(2): 156–168, 2012. ISSN 10715819. doi: 10.1016/j.ijhcs.2011.10.003.

[4] Ruth Sara Aguilar-Savén. Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129–149, jul 2004. ISSN 09255273. doi: 10.1016/S0925-5273(03)00102-6.

[5] Mack Alford. SREM at the age of eight; the distributed computing design system. *Computer*, 1985.

[6] A Antoniou and C Korakas. A trust-centered approach for building e-voting systems. In *EGOV'07 Proceedings of the 6th international conference on Electronic Government*, pages 366–377. Springer-Verlag, 2007.

[7] Ayrshire Civil Contingencies Team. Off-Site Contingency Plan (REDACTED VERSION). Technical report, Prepared for the West of Scotland Regional Resilience Partnership, 2015.

[8] RJ Badham, CW Clegg, and T Wall. Socio-technical Theory. In Waldemar Karwowski, editor, *International Encyclopedia of Ergonomics and Human Factors*. Taylor & Francis, 2001.

[9] Richard L Baskerville. Investigating information systems with action research. *Communications of AIS*, 2(3), 1999. ISSN 1529-3181.

[10] Richard L Baskerville and A Trevor Wood-Harper. A Critical Perspective on Action Research as a Method for Information Systems Research. *Journal of Information Technology*, 11, 1996.

[11] Gordon Baxter and Ian Sommerville. Socio-technical systems: From design methods to systems engineering. *Interacting with Computers*, 23(1):4–17, Jan 2011. ISSN 09535438. doi: 10.1016/j.intcom.2010.07.003.

[12] Gordon Baxter and Ian Sommerville. Evaluating emergency preparedness: Using responsibility models to identify vulnerabilities. In *Hostile Intent and Counter-Terrorism: Human Factors Theory and Application*. CRC Press, 2015.

[13] Gordon Baxter, Alan Burns, and Kenneth Tan. Evaluating timebands as a tool for structuring the design of socio-technical systems. In Philip D. Bust, editor, *Contemporary Ergonomics 2007: Proceedings of the International Conference on Contemporary Ergonomics*, 2007.

[14] Kent Beck and Martin Fowler. Bad Smells in Code. In *Refactoring: Improving the design of existing code*. Addison Wesley, 1999.

[15] TE Bell and TA Thayer. Software requirements: Are they really a problem? In *Proceedings of the 2nd international conference on Software engineering*, 1976.

[16] Peter Bishop and Robin Bloomfield. A methodology for safety case development. In *Proceedings of the Sixth Safety-critical Systems Symposium*, volume 2002, pages 1–10, 1998. ISBN 3540761896.

[17] Peter Bishop and Robin Bloomfield. A Methodology for Safety Case Development. *Safety and Reliability*, 20(1), 2000.

[18] AJC Blyth, J Chudge, John Dobson, and Ros Strens. ORDIT: a new methodology to assist in the process of eliciting and modelling organizational requirements. In *Proceedings of the Conference on Organizational Computing Systems, COOCS 1993*, pages 216–227, 1993.

[19] Jonathan Bowen and Michael Hinchey. Ten Commandments of Formal Methods ...Ten Years Later. *Computer*, 39(1):40–48, Jan 2006. ISSN 0018-9162. doi: 10.1109/MC.2006.35.

[20] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004. ISSN 1387-2532. doi: 10.1023/B:AGNT.0000018806.20944.ef.

[21] Lionel Briand, Victor Basili, Yong-Mi Kim, and Donald R Squier. A change analysis process to characterize software maintenance projects. In *Proceedings of the International Conference on Software Maintenance ICSM-94*, pages 38–49, 1994. ISBN 0-8186-6330-8. doi: 10.1109/ICSM.1994.336791.

[22] Lionel Briand, Walcelio Melo, Carolyn Seaman, and Victor Basili. Characterizing a Large-Scale Software and Assessing Organization. In *Proceedings of the 17th International Conference on Software Engineering*, pages 133–143, 1995. doi: 10.1145/225014.225027.

[23] British Standards Institution. *Reliability of systems equipment and components : Guide to failure modes, effects and criticality analysis*. 1991. ISBN 0580196607.

[24] Alan Burns and Gordon Baxter. Time bands in systems structure. In Denis Besnard, Cristina Gacek, and Cliff Jones, editors, *Structure for Dependability: Computer-Based Systems from an Interdisciplinary Perspective*. Springer-Verlag, 2006.

[25] M Burrows, M Abadi, and RM Needham. A Logic of Authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. ISSN 1364-5021. doi: 10.1098/rspa.1983.0054.

[26] Peter Checkland and Jim Scholes. *Soft Systems Methodology in Action*. Wiley, 1999. ISBN 0471986054.

[27] D Chen, B Vallespir, and G Doumeingts. GRAI integrated methodology and its mapping onto generic enterprise reference architecture and methodology. *Computers in Industry*, 33:387–394, 1997. ISSN 01663615. doi: 10.1016/S0166-3615(97)00043-2.

[28] Peter Pin-Shan Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976. doi: 10.1145/320434.320440.

[29] Laurence Cholvy, Frederic Cuppens, and Claire Saurel. Towards a logical formalization of responsibility. In *Proceedings of the 6th International Conference on Artificial Intelligence and Law*, pages 233–242, 1997.

[30] Jane Cleland-Huang. Don't fire the architect! Where were the requirements? *IEEE Software*, 31(2):27–29, 2014. ISSN 07407459. doi: 10.1109/MS.2014.34.

[31] Matthew Collinson and David Pym. Algebra and logic for resource-based systems modelling. *Mathematical Structures in Computer Science*, 19:959–1027, 2009. ISSN 0960-1295. doi: 10.1017/s0960129509990077.

[32] Matthew Collinson, Brian Monahan, and David Pym. Semantics for structured systems modelling and simulation. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, pages 34–43. ICST, 2010. ISBN 978-963-9799-87-5. doi: 10.4108/ICST.SIMUTOOLS2010.8631.

[33] George E Cooper, Maurice D White, and John K Lauber. Resource management on the flight deck. In *Proceedings of a NASA/Industry Workshop*, pages 31–58, 1980.

[34] Rita J Costello and Liu Dar-Biau. Metrics for Requirements Engineering. *Journal of Systems and Software*, 29(1):39–63, 1995.

[35] Kevin Crowston and Thomas W Malone. The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, 26(1):87–119, 1994.

[36] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1):3–50, 1993.

[37] Robert Darimont, Emmanuelle Delor, Philippe Massonet, and Axel van Lamsweerde. GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering. In *Proceedings of the 19th International Conference on Software Engineering*, pages 612–613, 1997.

[38] Robert Darimont, Emmanuelle Delor, Jean-Luc Roussel, and André Rifaut. Requirements Engineering with GRAIL / KAOS : Tell the Requirements, All the Requirements, and Nothing Else but the Requirements. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, page 299, 2002.

[39] Guy Dewsbury and John Dobson. *Responsibility and Dependable Systems*. Springer-Verlag, 2007. ISBN 978-1-84628-625-4. doi: 10.1007/978-1-84628-626-1.

[40] John Dobson. Responsibility Modelling: Basic Concepts. In *Responsibility and Dependable Systems*. Springer, 2007.

[41] John Dobson and David Martin. Enterprise Modeling based on Responsibility. In *Trust in Technology: A Socio-Technical Perspective*, pages 39–67. Springer, 2006.

[42] John Dobson and Mike Martin. Models for Understanding Responsibilities. In *Responsibility and Dependable Systems*, pages 115–129. Springer, 2007.

[43] John Dobson and Ian Sommerville. Roles Are Responsibility Relationships Really. 2005.

[44] Ed Downs, Peter Clare, and Ian Coe. *Structured Systems Analysis and Design Method*. Prentice Hall, 1988. ISBN 0-13-854324-0.

[45] James Druckman and Cindy Kam. Students as experimental participants: A defense of the "narrow data base". *Handbook of Experimental Political Science*, pages 41–57, 2011. ISSN 1556-5068. doi: 10.2139/ssrn.1498843.

[46] Helga Drummond. Riding a tiger: some lessons of Taurus. *Management Decision*, 36 (3):141–146, 1998. ISSN 0025-1747. doi: 10.1108/00251749810208922.

[47] Leticia Duboc, Emmanuel Letier, David S Rosenblum, and Tony Wicks. A Case Study in Eliciting Scalability Requirements. In *16th IEEE International Requirements Engineering Conference*, pages 247–252. IEEE, Sep 2008. ISBN 978-0-7695-3309-4. doi: 10.1109/RE.2008.22.

[48] Jordi Dunjo, Vasilis Fthenakis, Juan A Vilchez, and Josep Arnaldos. Hazard and operability (HAZOP) analysis. A literature review. *Journal of Hazardous Materials*, 173(1-3):19–32, 2010. ISSN 03043894. doi: 10.1016/j.jhazmat.2009.08.076.

[49] Eclipse Foundation. Eclipse Sirius, 2015. URL `https://eclipse.org/sirius/`. [Accessed on: 2016-05-29].

[50] Eclipse Foundation. Eclipse Modelling Framework, 2015. URL `http://www.eclipse.org/emf/`. [Accessed on: 2016-05-29].

[51] Eclipse Foundation. Acceleo, 2016. URL `http://www.eclipse.org/acceleo/`. [Accessed on: 2017-03-05].

[52] Elwyn Edwards. Man and Machine: Systems for Safety. In *Proceedings of British Airline Pilots Association Technical Symposium*, 1972.

[53] Eurocontrol. ACAS II Bulletin, 2007.

[54] Eurocontrol. Specification for Short Term Conflict Alert, 2007.

[55] Martin S Feather. Language support for the specification and development of composite systems. *ACM Transactions on Programming Languages and Systems*, 9(2):198–234, mar 1987. ISSN 01640925. doi: 10.1145/22719.22947.

[56] Massimo Felici. Capturing emerging complex interactions: Safety analysis in air traffic management. *Reliability Engineering and System Safety*, 91(12):1482–1493, 2006. ISSN 09518320. doi: 10.1016/j.ress.2006.01.010.

[57] Stephen Fenech, Gordon J Pace, and Gerardo Schneider. Automatic conflict detection on contracts. *Lecture Notes in Computer Science*, 5684:200–214, 2009. ISSN 03029743. doi: 10.1007/978-3-642-03466-4_13.

[58] Juliana K Filipe, Massimo Felici, and Stuart Anderson. Timed Knowledge-based Modelling and Analysis: On the Dependability of Socio-Technical Systems. In *Proceedings of the International Conference on Human Aspects of Advanced Manufacturing: Agility & Hybrid Automation*, pages 321–328, 2003.

[59] Anthony Finkelstein, Jeff Kramer, B Nuseibeh, L Finkelstein, and M Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):1–27, 1992.

[60] Flight Safety Foundation. Aviation Safety Network Accident Database, 2016. URL `https://aviation-safety.net/database/`. [Accessed on: 2016-05-21].

[61] Bent Flyvbjerg. Five Misunderstandings About Case-Study Research. *Qualitative Inquiry*, 12(2):219–245, 2006. ISSN 1077-8004. doi: 10.1177/1077800405284363.

[62] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999. ISBN 9780201485677.

[63] Robert Galliers. Choosing Appropriate Information Systems Research Approaches: A Revised Taxonomy. In *Information Systems Research: Contemporary Approaches and Emergent Traditions*. Elsevier, 1991.

[64] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison Wesley, 1994.

[65] Fred George. High-Altitude Upset Recovery. *Business and Commercial Aviation*, Jul 2011.

[66] Bill Gillham. *Research Interviewing*. Open University Press, 2005.

[67] Jaap Gordijn, Hans Akkermans, and Hans Van Vliet. Business modelling is not process modelling. In *Conceptual modeling for e-business and the web*, pages 40–51. Springer-Verlag, 2001.

[68] Jaap Gordijn, Eric Yu, and Bas Van Der Raadt. e-Service Design Using i* and e3value Modeling. *IEEE Software*, 23(3), 2006.

[69] Arnaud Gotlieb. TCAS Software Verification using Constraint Programming. *The Knowledge Engineering Review*, 00, 2009. ISSN 0269-8889.

[70] Richard Harper and William Newman. Designing for user acceptance using analysis techniques based on responsibility modelling. In *Conference Companion on Human factors in computing systems - CHI '96*, pages 217–218, New York, New York, USA, 1996. ACM Press. ISBN 0897918320. doi: 10.1145/257089.257289.

[71] Matthew Hause. The SysML Modelling Language. In *Proceedings of the Fifth European Systems Engineering Conference*, 2006.

[72] Stephen Hawking and Leonard Mlodinow. *The Grand Design*. 2010.

[73] William Heaven and Anthony Finkelstein. A UML profile to support requirements engineering with KAOS. *IEEE Proceedings - Software*, 151(1):10–27, 2004. ISSN 14625970. doi: 10.1049/ip-sen:20040297.

[74] H.W. Hendrick. Sociotechnical Systems Theory: the Sociotechnical Systems Model of Work Systems. In Waldemar Karwowski, editor, *International Encyclopedia of Ergonomics and Human Factors*. Taylor & Francis, 2001.

[75] Thomas Herrmann and Kai-Uwe Loser. Vagueness in models of socio-technical systems. *Behaviour & Information Technology*, 18(5):313–323, 1999. ISSN 0144-929X. doi: 10.1080/014492999118904.

[76] Erik Herzog and Asmus Pandikow. SysML - An Assessment. Technical report, 2005.

[77] Andrew Hilton, Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM: A Tool for Automatic Verification of Probabilistic Systems. *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 3920:441–444, 2006.

[78] C A R Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985. ISBN 0-13-153271-5.

[79] L. Hoepffner. Analysis of the HAZOP study and comparison with similar safety analysis systems. *Gas Separation and Purification*, 3(3):148–151, 1989. ISSN 09504214. doi: 10.1016/0950-4214(89)80027-1.

[80] Erik Hollnagel. *FRAM: the Functional Resonance Analysis Method*. Ashgate, 2012.

[81] Edward Huang, Randeep Ramamurthy, and Leon McGinnis. System and simulation modeling using SysML. In *Proceedings of the 2007 Winter Simulation Conference*, pages 796–803, 2007. ISBN 1424413060.

[82] M E Iacob, H Jonkers, M M Lankhorst, and H A Proper. Archimate 1.0 Specification. Technical report, The Open Group, 2009.

[83] ICAO. Annex 2 to the Convention on International Civil Aviation - Rules of the Air, 2005.

[84] ICAO. Airborne Collision Avoidance System (ACAS) Manual, 2006.

[85] Juhani Iivari and John Venable. Action Research and Design Science Research - Seemingly similar but decisively dissimilar. In *Proceedings of the 2009 European Conference on Information Systems*, 2009.

[86] InDeED Consortium. Responsibility Modelling of the Hunterston Nuclear Power Stations Off-Site Emergency Plan. Technical report, 2008.

[87] InDeED Consortium. InDeeD Hunterston Models and Notes. *GitHub repository*, 2008. URL `https://github.com/twsswt/indeed-hunterston-exercise-indus`.

[88] Isle of Anglesey County Council. Wylfa Nuclear Power Station Off-Site Emergency Plan, 2011.

[89] Daniel Jackson. Alloy: a lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology*, 11(2):256–290, Apr 2002. ISSN 1049331X. doi: 10.1145/505145.505149.

[90] Jonathon Jacky. *The Way of Z.* Cambridge University Press, 1997.

[91] Yosr Jarraya, Andrei Soeanu, Mourad Debbabi, and Fawzi Hassaine. Automatic verification and performance analysis of time-constrained SysML activity diagrams. In *Proceedings of the 14th Annual IEEE International Conference on the Engineering of Computer-Based Systems*, 2007. ISBN 0769527728.

[92] Pertti Jarvinen. Action research is similar to design science. *Quality and Quantity*, 41 (1):37–54, 2007. ISSN 00335177. doi: 10.1007/s11135-005-5427-1.

[93] Chris Johnson. *A Handbook of Incident and Accident Reporting*. Glasgow University Press, 2003.

[94] William Johnson. MORT: The Management Oversight and Risk Tree. *Journal of Safety Research*, 7(1):4–15, 1975.

[95] Cliff Jones. *Systematic software development using VDM*. Prentice Hall, second edition, 1990. ISBN 0-13-880733-7.

[96] T Jun, M A Piera, and J Nosedal. Analysis of induced traffic alert and collision avoidance system collisions in unsegregated airspace using a colored Petri net model. *Simulation*, 91(3):233–248, 2015. ISSN 00375497. doi: 10.1177/0037549715570357.

[97] Daniel Kahneman. *Attention and Effort*. Prentice Hall, 1973. ISBN 0130505188. doi: 10.2307/1421603.

[98] Tim Kelly and Simon Bates. The Costs, Benefits, and Risks Associated With Pattern-Based and Modular Safety Case Development. In *Proceedings of the UK MoD Equipment Safety Assurance Symposium*, 2005.

[99] Markus Kirchberg, Ove Sörensen, and Bernhard Thalheim. A BPMN Case Study: Paper Review and Submission System. In *GI Jahrestagung*, pages 4067–4081, 2009. ISBN 9783885792482.

[100] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2):105–112, 2009. ISSN 01674730. doi: 10.1016/j.strusafe.2008.06.020.

[101] George Koliadis and Aditya Ghose. Relating Business Process Models to Goal-Oriented Requirements Models in KAOS. In *Advances in Knowledge Acquisition and Management*, pages 25–39. Springer-Verlag, 2006.

[102] Gerald Kotonya and Ian Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley, 1998.

[103] John Krogstie. A Semiotic Approach to Quality in Requirements Specifications. In *Organizational Semiotics*. Springer, 2002. doi: 10.1007/978-0-387-35611-2_14.

[104] Philipe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, third edition, 2004.

[105] Kim G Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, 1997.

[106] Emmanuel Letier and William Heaven. Requirements modelling by synthesis of deontic input-output automata. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 592–601, may 2013. ISBN 978-1-4673-3076-3. doi: 10.1109/ICSE.2013.6606605.

[107] Emmanuel Letier, Jeff Kramer, Jeff Magee, and Sebastian Uchitel. Deriving event-based transition systems from goal-oriented requirements models. In *Automated Software Engineering*, volume 15, pages 175–206, may 2008. doi: 10.1007/s10515-008-0027-7.

[108] Nancy Leveson. A New Approach to Hazard Analysis for Complex Systems. In *Proceedings of the International Conference of the System Safety Society*, 2003.

[109] Nancy Leveson. A systems-theoretic approach to safety in software-intensive systems. *IEEE Transactions on Dependable and Secure Computing*, 1(1):66–86, Jan 2004. ISSN 1545-5971. doi: 10.1109/TDSC.2004.1.

[110] Nancy Leveson, Jon Damon Reese, and Mats P E Heimdahl. SpecTRM: A CAD System for Digital Automation. In *Proceedings of the Digital Avionics Systems Conference*, 1998.

[111] Nancy G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2011. ISBN 9780262016629. doi: 10.1017/CBO9781107415324.004.

[112] Lin Liu, Eric Yu, and John Mylopoulos. Security and privacy requirements analysis within a social setting. In *Proceedings of the 11th IEEE International Requirements Engineering Conference*, pages 151–161, 2003.

[113] Carolos Livadas, John Lygeros, and Nancy A. Lynch. High-level modeling and analysis of the traffic alert and collision avoidance system (TCAS). *Proceedings of the IEEE*, 88(7):926–947, 2000. ISSN 00189219. doi: 10.1109/5.871302.

[114] Russell Lock and Ian Sommerville. Modelling and Analysis of Socio-Technical System of Systems. In *Proceedings of the 2010 15th IEEE International Conference on Engineering of Complex Computer Systems*, pages 224–232. IEEE, 2010. ISBN 978-1-4244-6638-2. doi: 10.1109/ICECCS.2010.31.

[115] Russell Lock, Tim Storer, Ian Sommerville, and Gordon Baxter. Responsibility modelling for risk analysis. In *Reliability, risk and safety : theory and applications : Proceedings of the European Safety and Reliability Conference*, 2010.

[116] Russell Lock, Ian Sommerville, and Tim Storer. Responsibility Modelling. In *LSCITS Socio-Technical Systems Engineering Handbook*, pages 66–73. University of St. Andrews, 2011.

[117] Jonas Lundberg, Carl Rollenhagen, and Erik Hollnagel. What-You-Look-For-Is-What-You-Find: The consequences of underlying accident models in eight accident investigation manuals. *Safety Science*, 47(10):1297–1311, dec 2009. ISSN 09257535. doi: 10.1016/j.ssci.2009.01.004.

[118] Neil Maiden and Alistair G Sutcliffe. Requirements engineering by example: an empirical study. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, 1993. ISBN 0-8186-3120-1. doi: 10.1109/ISRE.1993.324828.

[119] Neil Maiden, Sara Jones, Cornelius Ncube, and James Lockerbie. Using i * in Requirements Projects : Some Experiences and Lessons. In *Social Modeling for Requirements Engineering*, pages 155–185. MIT Press, 2011.

[120] Mark Maier. Architecting principles for systems-of-systems. *Systems Engineering*, 1 (4), 1999.

[121] David Martin and Ian Sommerville. Patterns of Cooperative Interaction: Linking Ethnomethodology and Design. *ACM Transactions on Computer-Human Interaction*, 11(1):59–89, 2004. ISSN 10730516 (ISSN). doi: 10.1145/972648.972651.

[122] Raimundas Matulevičius and Patrick Heymans. Comparing Goal Modelling Languages: An Experiment. *Requirements Engineering: Foundation for Software Quality*, pages 18–32, 2007. ISSN 03029743. doi: 10.1007/978-3-540-73031-6_2.

[123] Jean McNiff and Jack Whitehead. *All you need to know about Action Research*. Sage, 2011. ISBN 9780857025838.

[124] JJC Meyer and RJ Wieringa. Applications of deontic logic in computer science: A concise overview. In *Deontic Logic in Computer Science: Normative System Specification*, pages 1–28. Wiley-Blackwell, 1993.

[125] JJC Meyer, RJ Wieringa, and FPM Dignum. The role of deontic logic in the specification of information systems. In *Logics for databases and information systems*, pages 17–44. John Wiley & Sons, 1998.

[126] Henry Mintzberg. Structure in 5'S: a Synthesis of the Research on Organization Design. *Management Science*, 26(3):322–341, 1980. doi: 10.1287/mnsc.26.3.322.

[127] Daniel Moody, Guttorm Sindre, Terje Brasethvik, and Arne Solvberg. Evaluating the quality of information models: empirical testing of a conceptual model quality framework. In *Proceedings of the 25th International Conference on Software Engineering*, pages 295–305, 2003. ISBN 076951877X. doi: 10.1109/ICSE.2003.1201209.

[128] Michael Muehlen and Danny T Ho. Service Process Innovation: A Case Study of BPMN in Practice. In *Proceedings of the 41st Hawaii International Conference on System Sciences*, 2008. ISBN 0-7695-3075-8. doi: 10.1109/HICSS.2008.388.

[129] Enid Mumford. The story of socio-technical design: Reflections on its successes, failures and potential. *Information Systems Journal*, 16(4):317–342, 2006. ISSN 13501917. doi: 10.1111/j.1365-2575.2006.00221.x.

[130] Tadao Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. ISSN 15582256. doi: 10.1109/5.24143.

[131] Hiroyuki Nakagawa, Kenji Taguchi, and Shinichi Honiden. Formal Specification Generator for KAOS. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated Software Engineering*, pages 531–532, 2007. ISBN 9781595938824.

[132] Thomas H Naylor, J M Finger, James L McKenney, William E Schank, and Charles C Holt. Verification of Computer Simulation Models. *Management Science*, 14(2), 1967.

[133] North Ayrshire Council. Hunterston Nuclear Power Stations Off-Site Emergency Plan, 2006.

[134] Briony J Oates. *Researching Information Systems and Computing*. Sage, 2006. ISBN 9781412902243.

[135] Object Management Group. MOF Core Specification, 2013.

[136] Samir Okasha. *Philiosophy of Science: A Very Short Introduction*. Oxford University Press, 2016.

[137] Chun Ouyang, Marlon Dumas, Arthur HM Ter Hofstede, and Wil MP Van Der Aalst. From BPMN process models to BPEL Web services. In *Proceedings of ICWS 2006: 2006 IEEE International Conference on Web Services*, pages 285–292, 2006. ISBN 0769526691. doi: 10.1109/ICWS.2006.67.

[138] Vineet Padmanabhan, Guido Governatori, Shazia Sadiq, Robert Colomb, and Antonino Rotolo. Process modelling: the deontic way. In *Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling*, 2006.

[139] Terence Parr. ANTLR, 2015.

[140] Oscar Pastor, Hugo Estrada, and Alicia Martinez. Strengths and Weaknesses of the i* Framework. In *Social Modeling for Requirements Engineering*. MIT Press, 2011.

[141] Ken Peffers, Tuure Tuunanen, Charles E Gengler, Matti Rossi, Wendy Hui, Ville Virtanen, and Johanna Bragge. The Design Science Research Process: A Model for Producing and Presenting Information Systems Research. In *Proceedings of Design Research in Information Systems and Technology (DESRIST'06)*, pages 83–106, 2006.

[142] A Platt and S Warwick. Review of soft systems methodology. *Industrial Management & Data Systems*, 95(4):19–21, 1995. ISSN 0263-5577. doi: 10.1108/02635579510086698.

[143] Karl Popper. *The Logic of Scientific Discovery*. Hutchinson, 1959. ISBN 0415278449. doi: 10.1016/S0016-0032(59)90407-7.

[144] Ben Potter, Jane Sinclair, and David Till. *An Introduction to Formal Specification and Z*. Prentice Hall, 1996.

[145] Amy Pritchett. Aviation Automation : General Perspectives and and Alerts. *Reviews of Human Factors and Ergonomics*, pages 82–113, 2009. doi: 10.1518/155723409X448026.

[146] Amy Pritchett, Elizabeth Fleming, William Cleveland, Vlad Popescu, Dhruv Thakkar, and Jonathon Zoetrum. Pilot's Information Use During TCAS Events, and Relationship to Compliance to TCAS Resolution Advisories. In *Proceedings of the Human Factors and Ergonomics Society 56th Annual Meeting*, pages 26–30, 2012. ISBN 9780945289418. doi: 10.1177/1071181312561026.

[147] Charles C Ragin. "Casing" and the process of social inquiry. In *What is a case? Exploring the foundations of social inquiry.*, pages 217–227. Cambridge University Press, 1992. ISBN 0521421888.

[148] Devina Ramduny-Ellis and Alan Dix. Modelling in Practice. In *Responsibility and Dependable Systems*. Springer, 2007.

[149] Jan Recker, Marta Indulska, Michael Rosemann, and Peter Green. How good is BPMN really? Insights from Theory and Practice. In *Proceedings of the 14th European Conference on Information Systems*, pages 1046–1072, 2006. ISBN 0798855444.

[150] Respect-IT SA. Objectiver Technical Datasheet, 2012.

[151] GC Roman. A taxonomy of current issues in requirements engineering. *Computer*, 18 (4):14–23, 1985.

[152] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual*. Addison-Wesley, second edition, 2004.

[153] Paul M Salmon, Guy H Walker, and Neville A Stanton. Pilot error versus sociotechnical systems failure: a distributed situation awareness analysis of Air France 447. *Theoretical Issues in Ergonomics Science*, 17(1):64–79, 2016.

[154] Robert G Sargent. Verification and Validation of Simulation Models. In S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, editors, *Proceedings of the 2003 Winter Simulation Conference*, 2003. ISBN 0-7803-8131-9. doi: 10.1109/WSC.2003.1261400.

[155] Ken Schwaber and Mike Beedle. *Agile Software Development with SCRUM*. Prentice Hall, 2001.

[156] Ben Shneiderman. Science 2.0. *Science*, 319(5868):1349–1350, 2008. ISSN 0036-8075. doi: 10.1126/science.1153539.

[157] Robbie Simpson. Responsibility Modelling Editor. *GitHub repository*, 2015. URL `https://github.com/twsswt/resme`.

[158] Robbie Simpson. Hunterston Responsibility Models. *GitHub repository*, 2015. URL `https://github.com/twsswt/resme-hunterston-case-study`.

[159] Robbie Simpson. TCAS Responsibility Models. *GitHub repository*, 2016. URL `https://github.com/twsswt/resme-tcas-case-study`.

[160] Robbie Simpson. Domain Expert Evaluations of a TCAS Responsibility Model, 2016. URL `http://researchdata.gla.ac.uk/372/`.

[161] Robbie Simpson. Responsibility Modelling User Study Materials. *GitHub repository*, 2016. URL `https://github.com/robmods/resme-user-study`.

[162] Robbie Simpson and Tim Storer. Formalising Responsibility Modelling for Automatic Analysis. *Lecture Notes in Business Information Processing*, 231:125–140, 2015. doi: 10.1007/978-3-319-24626-010.

[163] Robbie Simpson and Tim Storer. Responsibility Model Editor update site, 2016. URL `http://www.dcs.gla.ac.uk/{~}tws/software/resme-update-site/`.

[164] Dag I K Sjoberg, Jo E Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, Nils Kristian Liborg, and Anette C Rekdal. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, 31(9): 733–753, 2005. ISSN 00985589. doi: 10.1109/TSE.2005.97.

[165] Gideon Sjoberg. Contradictory Functional Requirements and Social Systems. *Journal of Conflict Resolution*, 4(2):198–208, 1960.

[166] Ian Sommerville. Models for Responsibility Assignment. In *Responsibility and Dependable Systems*. Springer, 2007.

[167] Ian Sommerville. Causal responsibility models. In *Responsibility and Dependable Systems*. Springer, 2007.

[168] Ian Sommerville, Tim Storer, and Russell Lock. Responsibility Modelling for Contingency Planning. 2007.

[169] Ian Sommerville, Russell Lock, Tim Storer, and John Dobson. Deriving information requirements from responsibility models. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering*, pages 515–529, 2009.

[170] Ian Sommerville, Tim Storer, and Russell Lock. Responsibility modelling for civil emergency planning. *Risk Management*, 11(3-4):179–207, Jul 2009. ISSN 1460-3799. doi: 10.1057/rm.2009.11.

[171] JM Spivey. *The Z Notation*. Prentice Hall, 1989.

[172] Tim Storer. Observations of Exercise Indus. 2007.

[173] Tim Storer. Observations of Excerise Kilchattan. 2008.

[174] Tim Storer. Hunterston responsibility model. 2008.

[175] Tim Storer and Russell Lock. An integrated model of responsibility for the analysis of the dependability of socio-technical systems. 2007.

[176] Tim Storer and Russell Lock. Modelling Responsibility. 2009.

[177] Tim Storer, Karen Renaud, and William Bradley Glisson. Patterns of Information Security Postures for Socio-Technical Systems and Systems-of-Systems. In *Proceedings of the First International Workshop on Cyberpatterns*, pages 30–34, 2012.

[178] Ros Strens and John Dobson. How responsibility modelling leads to security requirements. In *Proceedings of the 16th National Computer Security Conference*, pages 143–149, 1993.

[179] Ros Strens and John Dobson. Responsibility modelling as a technique for organisational requirements definition. *Intelligent Systems Engineering*, 3(1):20–26, 1994. ISSN 09639640. doi: 10.1049/ise.1994.0003.

[180] Markus Strohmaier, Jennifer Horkoff, and Eric Yu. Can patterns improve i* modeling? two exploratory studies. In *Requirements Engineering: Foundation for Software Quality*, pages 153–167. Springer, 2008.

[181] Alistair G Sutcliffe. Requirements analysis for socio-technical system design. *Information Systems*, 25(3):213–233, 2000. ISSN 03064379. doi: 10.1016/S0306-4379(00)00016-8.

[182] CD Swann and ML Preston. Twenty-five years of HAZOPs. *Journal of loss prevention in the Process Industries*, 8(6):349–353, 1995.

[183] Joseph C Sweeney. ARCO Chemical's HAZOP Experience. *Process Safety Progress*, 12(2):83–91, 1993.

[184] Eric Trist. The Evolution of socio-technical systems. In *Proceedings of the Conference on Organizational Design and Performance*, volume 2, pages 1–67, 1981. doi: 0-7743-6286-3.

[185] UK Parliament. Civil Contingencies Act 2004, 2004.

[186] Peter Underwood and Patrick Waterson. A Critical Review of the STAMP , FRAM and Accimap Systemic Accident Analysis Models. In Neville A . Stanton, editor, *Advances in Human Aspects of Road and Rail Transportation*. CRC Press, 2012.

[187] Axel Van Lamsweerde. Requirements Engineering in the Year 2000: A Research Perspective. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 5–19, 2000. ISBN 1-58113-206-9. doi: 10.1145/337180.337184.

[188] Axel van Lamsweerde, Robert Darimont, and Philippe Massonet. The Meeting Scheduler System - Preliminary Definition. Working Paper, 1993. URL `http://www.cs.cmu.edu/{~}ModProb/CSdef1.html`. [Accessed on: 2017-05-19].

[189] Axel van Lamsweerde, Robert Darimont, and Philippe Massonet. Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. In *Proceedings of 1995 IEEE International Symposium on Requirements Engineering (RE'95)*, pages 194–203. IEEE Comput. Soc. Press, 1995. ISBN 0-8186-7017-7. doi: 10.1109/ISRE.1995.512561.

[190] WE Vesely, FF Goldberg, NH Roberts, and DF Haasl. Fault tree handbook, 1981.

[191] Jos Warner and Anneke Kleppe. *The Object Constraint Language: Precise Modelling with UML*. Addison-Wesley, 1998.

[192] Kun Wei, Jim Woodcock, and Alan Burns. Timed Circus: Timed CSP with the miracle. In *2011 16th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2011*, pages 55–64, 2011. ISBN 9780769543819. doi: 10.1109/ICECCS.2011.13.

[193] Kun Wei, Jim Woodcock, and Alan Burns. Modelling Temporal Behaviour in Complex Systems with Timebands. In Mike Hinchey and Lorcan Coyle, editors, *Conquering Complexity*, pages 277–307. Springer London, London, 2012. ISBN 978-1-4471-2296-8. doi: 10.1007/978-1-4471-2297-5.

[194] J D Welch and V A Orlando. Active Beacon Collision Avoidance System (BCAS): Functional Overview, 1980.

[195] Stephen A White. Introduction to BPMN. *Business Process Trends*, 2004.

[196] RD Williams. Managing the Development of Reliable Software. In *Proceedings of the international conference on Reliable software*, 1975.

[197] Richard Winter. *Action-Research and the Nature of Social Inquiry*. Avebury, 1987. ISBN 0566055325.

[198] R Wirfs-Brock and Brian Wilkerson. Object-oriented design: a responsibility-driven approach. In *Proceedings of the 1989 ACM OOPSLA conference on object-oriented programming*, pages 71–75, 1989. ISBN 0897913337.

[199] Claes Wohlin, Per Runeson, Martin Host, Magnus C Ohlsson, Bjorn Regnell, and Andrews Wesslen. *Experimentation in Software Engineering*. Springer, 2013. ISBN 9788578110796. doi: 10.1017/CBO9781107415324.004.

[200] Robert K Yin. *Case Study Research: Design and Methods*. Sage, fourth edition, 2009. ISBN 978-1-4129-6099-1.

[201] Eric Yu. Modelling Organizations for Information Systems Requirements Engineering. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 34–41, 1993. ISBN 0818631201. doi: 10.1109/ISRE.1993.324839.

[202] Eric Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, 1995.

[203] Eric Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of ISRE '97: 3rd IEEE International Symposium on Requirements Engineering*, pages 226–235. IEEE Comput. Soc. Press, 1997. ISBN 0-8186-7740-6. doi: 10.1109/ISRE.1997.566873.

[204] Eric Yu and Lin Liu. Modelling Trust for System Design Using the i * Strategic Actors Framework. In *Trust in Cyber-Societies*, pages 175–194. Springer, 2001.

Appendix A

User Study Materials

The Meeting Scheduler System Definition

Scheduling Meetings:

Meetings are typically arranged in the following way. A *meeting initiator* asks all potential meeting attendees for the following information based on their personal agenda:

- a set of dates on which they cannot attend the meeting (hereafter referred as exclusion set);
- a set of dates on which they would prefer the meeting to take place (hereafter referred as preference set).

A *meeting date* is defined by a pair (calendar date, time period). The exclusion and preference sets are contained in some time interval prescribed by the meeting initiator (hereafter referred as *date range*).

The initiator also asks active participants to provide any special equipment requirements on the meeting location (e.g., overhead-projector, workstation, network connection, telephones, etc.); he/she may also ask important participants to state preferences about the meeting location.

The proposed meeting date should belong to the stated date range and to none of the exclusion sets; furthermore it should ideally belong to as many preference sets as possible. A date conflict occurs when no such date can be found. A conflict is strong when no date can be found within the date range and outside all exclusion sets; it is weak when dates can be found within the date range and outside all exclusion sets, but no date can be found at the intersection of all preference sets. Conflicts can be resolved in several ways:

- the initiator extends the date range;
- some participants remove some dates from their exclusion set;
- some participants withdraw from the meeting;
- some participants add some new dates to their preference set.

A meeting room must be available at the selected meeting date. It should meet the equipment requirements; furthermore it should ideally belong to one of the locations preferred by as many important participants as possible. A new round of negotiation

may be required when no such room can be found.

The meeting initiator can be one of the participants or some representative (e.g., a secretary).

System Requirements

The purpose of the *meeting scheduler system* is to support the organization of meetings - that is, to determine, for each meeting request, a meeting *date* and *location* so that most of the intended participants will effectively participate. The meeting date and location should thus be as convenient as possible to all participants. Information about the meeting should also be made available as early as possible to all potential participants. The intended system should considerably reduce the amount of overhead usually incurred in organizing meetings where potential attendees are distributed over many different places. On another hand, the system should reflect as closely as possible the way meetings are typically managed (see the domain theory above).

The system should assist users in the following activities.

- Plan meetings under the constraints expressed by participants
- Replan a meeting dynamically to support as much flexibility as possible. On one hand, participants should be allowed to modify their exclusion set, preference set and/or preferred location before a meeting date/location is proposed. On the other hand, it should be possible to take some external constraints into account after a date and location have been proposed - e.g., due to the need to accommodate a more important meeting. The original meeting date or location may then need to be changed; sometimes the meeting may even be cancelled. In all cases some bound on replanning should be set up.
- Support conflict resolution according to resolution policies stated by the client.
- Manage all the interactions among participants required during the organization of the meeting - to communicate requests, to get replies even from participants not reacting promptly, to support the negotiation and conflict resolution processes, to make participants aware of what's going on during the planning process, to keep participants informed about schedules and their changes, to make them confident about the reliability of the communications, etc.
- Keep the amount of interaction among participants (e.g., number and length of

messages, amount of negotiation required) as small as possible.

The meeting scheduler system must in general handle several meeting requests *in parallel*. Meeting requests can be competing by overlapping in time or space. Concurrency must thus be managed.

The following aspects should also be taken into account.

- The system should accommodate decentralized requests; any authorized user should be able to request a meeting independently of his whereabouts.
- Physical constraints may not be broken - e.g., a person may not be at two different places at the same time, a meeting room may not be allocated to more than one meeting at the same time.
- The system should provide an appropriate level of performance, for example:
  - the elapsed time between the submission of a meeting request and the determination of the corresponding meeting date/location should be as small as possible;
  - the elapsed time between the determination of a meeting date/location and the communication of this information to all participants concerned should be as small as possible;
  - a lower bound should be fixed between the time at which the meeting date is determined and the time at which the meeting is actually taking place.
- Privacy rules should be enforced; a non-privileged participant should not be aware of constraints stated by other participants.
- The system should be usable by non-experts.
- The system should be customizable to professional as well as private meetings. These two modes of use are characterized by different restrictions on the time periods that may be allocated (e.g., meetings during office hours, private activities during leisure time).
- The system should be flexible enough to accommodate evolving data - e.g., the sets of concerned participants may be varying, the address at which a participant can be reached may be varying, etc.

Responsibility Modelling Tutorial

Responsibility modelling is a notation and methodology for modelling complex socio-technical systems - systems that combine human, organisational and technical elements. Responsibility models consist of actors, responsibilities and resources (collectively, we call these entities). We use responsibilities as the core element of these models because we believe that they are a more intuitive way of capturing real-world activities than goal or process models.

Responsibilities are tasks, duties and obligations and can range from very abstract moral or legal principles to specific details about a particular task or process. The aim of the system is to successfully complete (discharge) all the responsibilities, which may require the actions of actors or the use of resources. Both resources and actors are flexible - resources can be physical materials or pieces of information, and actors can be organisations, individuals or technical systems like computer programs or sensors.

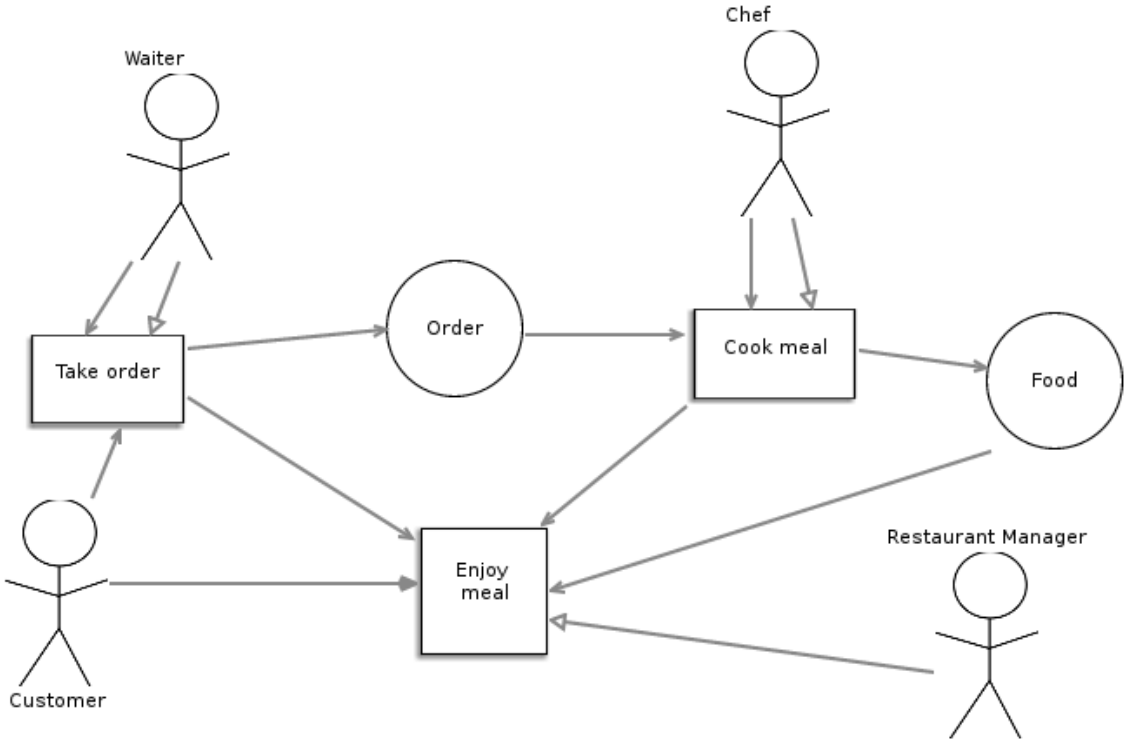
A responsibility model consists of these three entity types, linked by relations. Relations are:

- Required actor/resource/responsibility: indicates that the required entity is necessary to discharge the source responsibility
- Production: indicates that a resource is produced by the successful discharge of a responsibility
- Holds: indicates that an actor is responsible for / accountable for a particular responsibility (This contrasts with required actor, which just indicates the actor is needed)

It is also important to note that responsibility models are stateless, unlike many similar notations. Responsibilities are either discharged or are not discharged, resources are produced or not produced etc. over the entire timeframe of the system, rather than in discrete steps. This simplifies the timing of a system - for example, we are not concerned how quickly a responsibility is discharged - it is simply either successful (and hence ready on time) or unsuccesful.

In our graphical representation, responsibilities are represented as rectangles, actors as stickmen and resources as ovals. Relations are indicated by directional arrows with filled arrowheads, with the exception of the Holds relation, which is a directional arrow with a open arrowhead. There is no specific notation for each type of relation - the type is inferred from the source and target entities.

Consider this example of ordering and eating food in a restaurant:



Here we have three responsibilities (Enjoy meal, Take order, Cook meal), four actors (Waiter, Customer, Chef, Restaurant Manager) and two resources (Order, Food). We can see that the customer and the waiter are both required in order to take the order, but only the waiter is accountable for it. The chef is both required for and is accountable for cooking the meal, which produces food as long as the order is available. The meal can be enjoyed as the order has been taken, the meal cooked, the food is available and the customer is available - the manager is not directly involved in any way, but is responsible for ensuring that it is successfully discharged.

There are lots of different techniques for creating responsibility models from source materials. We often use a system of entity detection similar that used for ER diagrams - read through source documents and identify the actors, responsibilities and resources and create them in the model, before making a second pass of the documents to determine the relations. Organisations and roles correspond to actors, while pieces of information correspond to resources and tasks, obligations etc. correspond to responsibilities. Of course, many other elicitation strategies are perfectly suitable.

We have developed a toolkit for responsibility modelling called RESME, which is available on GitHub: **github.com/twsswt/resme**

For each of installation, we recommend first making a clean install of the ObeoDesigner Eclipse build: **obeodesigner.com/en** which contains all the necessary dependencies. If you have any problems, do email us!