Formalising Responsibility Modelling for Automatic Analysis

Robbie Simpson and Tim Storer

University of Glasgow

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. Responsibility modelling is formalised as a structured approach for socio-technical system requirements specification and modelling, with well-defined semantics and support for automated structure and validity analysis. The effectiveness of the approach is demonstrated by two case studies of software engineering methodologies.

1.1 Introduction

A range of research efforts have highlighted the importance of understanding the social context that a technical software system is developed for [24]. Baxter and Sommerville [2] argues that a common cause of failure in software development is the focus on the functional requirements for the system at hand, to the exclusion of an understanding of the wider organisation(s) in which the system is to be employed. Besnard and Baxter [3] observes that this focus can limit the ability of users to recover a system when technical components fail. Even when the importance of the wider organisational context is understood, the methods available to development teams are inadequate in the face of the scale of modern systems engineering challenges [10].

As a consequence, a variety of research efforts have been undertaken to develop methods that enable the modelling, analysis and construction of large scale systems comprised of human, organisational and technical components. Such systems are often referred to as *socio-technical* systems [31]. In particular, *responsibilities* have been proposed by a variety of research efforts as a suitable abstraction for capturing the structure of socio-technical systems [4, 8, 9, 11]. Sommerville et al. [29] provided a working definition of a responsibility as:

> "A duty, held by some agent, to achieve, maintain or avoid some given state, subject to conformance with organisational, social and cultural norms."

Responsibilities therefore provide a different type of abstraction from other approaches that have been proposed for modelling socio-technical-systems, such as goal [18] or activity [1] oriented approaches. Describing a socio-technical system in terms of responsibilities is concerned with monitoring and managing some part of the state of a socio-technical system, rather than the performance of specific task or activity. Similar to goal-oriented approaches, responsibilities can be discharged (achieved) in a variety of different ways by a variety of different actors (whether human, technical, or organisational).

However, responsibilities are distinct from goals, because of the incorporation of the notion of *duty* or *commitment* by the agent to discharge the responsibility. Goal oriented approaches assume that if an agent has the necessary capabilities to achieve a goal then it will succeed. However, a responsibility oriented approach acknowledges the greater complexities of socio-technical systems that may mean a responsibility is not discharged, even if an actor has the capability to do so. In this situation, it is possible to model fall back mechanisms, such as consequences for the agent, or 'backup' responsibilities held by other agents.

In addition, Sommerville et al. [27] argues that responsibilities are easier to elicit from a problem domain through discussion with stakeholders than other abstractions, such as goals. Stakeholders working within a system may struggle to explain the goals associated with their work without the description sounding somewhat artificial: "My goal is to secure the building at night." Describing the same work in terms of responsibilities is more intuitive: "I am responsible for securing the building at night."

Responsibility modelling has been shown to be effective as a basis for analysing a variety of socio-technical systems engineering activities, including analysing organisational roles [9]; risk analysis [22]; capturing security policies [30]; and requirements engineering [28]. However, much of this work depends on manual intervention and analysis by the engineer. As a consequence, it is difficult to use these methods for the construction of proposed systems. They may also be prove too costly when used for more complex systems involving many responsibilities.

This paper introduces an extension to the current state of the art in responsibility modelling. The novel contribution is a formalisation of the graphical responsibility notation adopted by Lock and Sommerville [21], Sommerville et al. [29, 28, 27]. The paper demonstrates how a formalisation of the notation as a language can enhance the automation of analytical techniques and aid in the construction of socio-technical system designs.

Section 1.2 reviews previous work employing responsibility modelling for analysing socio-technical systems, as well as other approaches, in greater detail. Section 1.3 presents our formalisation of the responsibility modelling notation used in previous research. The notation is formalised as a declarative language about the responsibilities held in a socio-technical system; the agents that hold those responsibilities; and the resources required or produced as a result of responsibility discharge.

Section 1.4 presents and analyses two example responsibility models, based on standard models of software development team organisation. The example illustrates how the application of a formal approach to responsibility modelling can automatically elicit system level vulnerabilities without requiring expert analysis.

Section 1.5 examines the current tool support for formalised responsibility modelling. A prototype responsibility modelling software tool is introduced, as well as several different techniques for automated model analysis.

Finally, Section 1.6 summarises the paper, emphasising difficulties of effectively modelling social-technical systems and highlighting the strengths of formalised responsibility modelling as a semi-formal approach that combines flexibility with sufficient rigour for automation.

1.2 Modelling Socio-Technical Systems

Developing modelling techniques for socio-technical systems requires a difficult trade-off between formal notations that support automated analysis and provide input for construction techniques such as refinement or static verification; and notations that enable validation by end-users and other stakeholders of a proposed system. An additional challenge is provide techniques and notations that are scalable so that a system can be described consistently at several different levels of granularity, while also managing the complexity and nuances that are often encountered when modelling socio-technical systems.

Systems modelling techniques such as SysML [16], the Department of Defense Architecture Framework (DoDAF) [17] and The Open Group Architecture Framework (TOGAF) [14] have been developed to support larger scale systems engineering. SysML is an extension of the Unified Modelling Language (UML) that is intended for capturing larger scale concerns and support wider systems engineering efforts that are not addressed by the UML. A particular example is closer integration of non-functional requirements into design concerns.

Leveson [19], Leveson and Dulac [20] describe an alternative methodology, STAMP, for analysing interactions between heterogeneous systems components. In addition to providing a modelling notation based on systems dynamics, the technique also incorporates a methodology for identifying potential instabilities in system behaviour that may eventually lead to failure. Leveson illustrates the technique by re-constructing the causes of the failure of the MilStar Satellite launch in 1999 [19]. A disadvantage is the need for expert application of the method to identify weaknesses.

Deontic logic provides a formal basis for reasoning about *norms* in socio-technical systems, employing operators such as *commitments* [23]. Several authors have proposed methods for capturing different aspects of socio-technical system specification and design based on deontic logic. For example: Garion and van der Torre [13] proposed a design-by-contract style language for multi-agent systems based on deontic logic; Padmanabhan et al. [25] proposed the integration of deontic logic concepts into business process modelling methods and Cholvy et al. [6] proposed the use of deontic logic as means of formalising concepts of responsibility. However, we are unaware of work that bridges the gap between formal expressions of system specifications in a deontic logic and notations that can be comprehended by non-expert domain stakeholders.

Goal oriented approaches [18], such as KAOS [7] and i* [32] employ a graphical notation to capture the goal seeking behaviours of agents in a system. These techniques provide a means for decomposing high level system goals into sub-goal structures to be pursued by actors within a socio-technical system.

Responsibility modelling has been proposed as a means of understanding a variety of different concerns in socio-technical systems design. Blyth et al. [4] first proposed the modelling and analysis of responsibilities in the Ordit methodology. The aim was to identify and understand organisational requirements for socio-technical systems. Later, Harper and Newman [15] proposed employing a structured language approach to defining responsibilities. Although the only outlined the approach, it shows how responsibility modelling can be useful in identifying conflicts between different actor-roles in an organisation.

Strens and Dobson [30] first proposed the use of responsibility modelling as a means of better understanding the appropriate configuration of security policies in an organisation. Later, Feltus and Petit [12] proposed a formalisation of responsibility in an organisational context as a means of modelling and reasoning about organisational policies. Feltus et al. [11] extended this work by developing a methodology for modelling organisational policies using responsibilities as a formalism.

In early work in the DIRC[1] project, Dobson and Sommerville [9] showed the use of responsibility models as a means of capturing the different roles undertaken in a socio-technical system. Sommerville et al. [29] applied the responsibility modelling techniques developed in the DIRC project to contingency planning for civil emergencies. The analysis was used to capture the responsibility vulnerabilities that caused some of the disruption during the response to a major civil incident. The paper demonstrated that responsibility modelling could be applied to an inter-organisational view of a socio-technical system.

Lock and Sommerville [21] also applied responsibility modelling to understanding the evolution of larger scale systems and systems. The work showed that responsibility modelling could be applied to the modelling, analysis and understanding of risks of collections of systems that are loosely coordinated for some larger purpose, but remain under autonomous control of independent organisations.

As noted by Baxter and Sommerville [2], less consideration has been given to the methods and notations which support the engineering of socio-technical systems by providing information of use to a systems engineer.

Sommerville et al. [28] began to explore this challenge by using responsibility models as a means of deriving early stage information systems requirements. The technique derived requirements based on the likely information needs of an actor discharging a responsibility. These requirements can then be used to

[1] Dependability Interdisciplinary Research Collaboration

support the configuration of information systems based on enterprise resource planning frameworks, rather than the complete development of new systems. This approach was illustrated by developing information requirements for an inter-organisational emergency management system.

In general, previous work on responsibility modelling did not deliver the unified approach that was hoped for. The graphical representation and underlying primitives varied between different works; some papers used a large set of entities, while others paired this down to a common core. Tool support was limited, and as a result analysis often had to be performed by hand.

1.3 Formalism

1.3.1 Notation

Responsibility modelling uses a simple notation, consisting of responsibilities, actors and resources as well as the relationships between them. These elements can either be represented graphically or textually, and tools to convert from one to the other are under development. Figure 1.1 provides a simple example of the graphical notation.

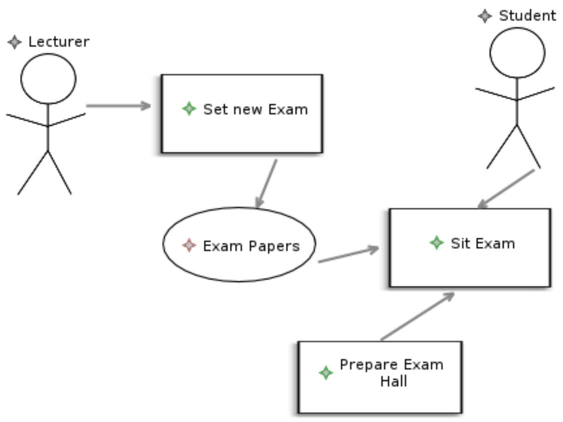


Fig. 1.1. Sample responsibility model for a university exam. The exam papers are produced and consumed; the Sit Exam responsibility is dependent on both the actor Student and the responsibility Prepare Exam Hall.

Responsibilities Responsibilities are duties that must be discharged by the system. They are the core of responsibility modelling, with all other artefacts either discharging these responsibilities; or resources that are produced by or provide support that enables the responsibility to be discharged. Responsibilities can be specified at any level of detail from high level abstractions to specific implementation approaches.

Actors Actors are entities within the system that act to discharge responsibilities. Actors can either be human, technical or organisational. Human actors are roles within the system (rather than individuals) and technical actors are automated processes such as software daemons or machinery. The different types of actors are treated homogeneously in the notation: all actor types can be assigned to responsibilities, which indicates they have a responsibility to discharge them.

Resources Resources represent objects that are required in order for a responsibility to be discharged, such as documents required to complete a managerial task or physical equipment required to complete a practical task. Responsibilities cannot be discharged unless the appropriate resources are available, and resources can either be produced by other responsibilities or can be initialised as already existing.

1.3.2 Properties & Relations

Assignment Actors are assigned to responsibilities, indicating that the actor holds a duty to discharge that particular responsibility. In the standard case the responsibility can be successfully discharged (assuming all other requirements are met) as long as at least one actor assigned to it is capable, although more complex behaviour can be modelled using constraints (see Section 1.5).

Decomposition & Delegation Responsibilities can be decomposed into sub-responsibilities that retain a relationship to their parent. Decomposition allows for refinement of the model, and allows detailed system models to be produced and linked back to more abstract designs. When responsibilities are decomposed in this way they may not be discharged unless all sub-responsibilities have also been discharged.

Decomposition of responsibilities also introduces a related property, that of delegation. If an actor is assigned to a high-level responsibility that is decomposed into sub-responsibilities there is an implied assignment to those sub-responsibilities as well. However, these sub-responsibilities can also be assigned to different actors, in which case the responsibility to discharge is delegated.

Supervision When responsibilities are delegated the original actor no longer plays an active role in discharging those responsibilities. However, they are still responsible for the high-level responsibility, which cannot be discharged without also discharging the delegated responsibilities. This creates an implicit relationship of supervision between the actor assigned to the high level goal and the actors that responsibilities have been delegated to. This enables responsibility modelling to generate a model of the organisation hierarchy within the model, despite this not being explicitly specified. In particular, this approach can highlight when the in-practice hierarchy implied in the system design is different from the formal managerial structure.

Scope Generally in responsibility modelling the scope of the system is implicit, relying on those interpreting the model to understand how the system sits in the wider context. However this approach is subjective, and can easily lead to ambiguity, especially when refining from a high level model down to an implementation design. Therefore objects on the edge of the system scope can be explicitly indicated. These objects should not be further refined, and can be considered to operate as black boxes. They are particularly useful when indicating the boundary between the system currently modelled and another, interacting social-technical system.

1.3.3 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 system analysis level the responsibility model should aim for completeness. By this stage of development all elements of the system should have been decided upon, 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 tasks of the system.

1.4 Modelling

This section illustrates the use of responsibility models in formally analysing the structure of socio-technical systems. For motivating examples, two software development team models are considered: the surgical team described by Brooks [5] and the Scrum agile team structure described by Schwaber and Beedle [26].

1.4.1 The Surgical Team

The surgical team is a model for small-scale software development. Figure 1.2 illustrates the surgical team using the responsibility modelling notation described above.

The model proposes a team structure consisting of ten roles, two of which are core developers (the surgeon/chief developer and the co-pilot/junior development) with the rest comprising a support staff. The surgical team is a not a

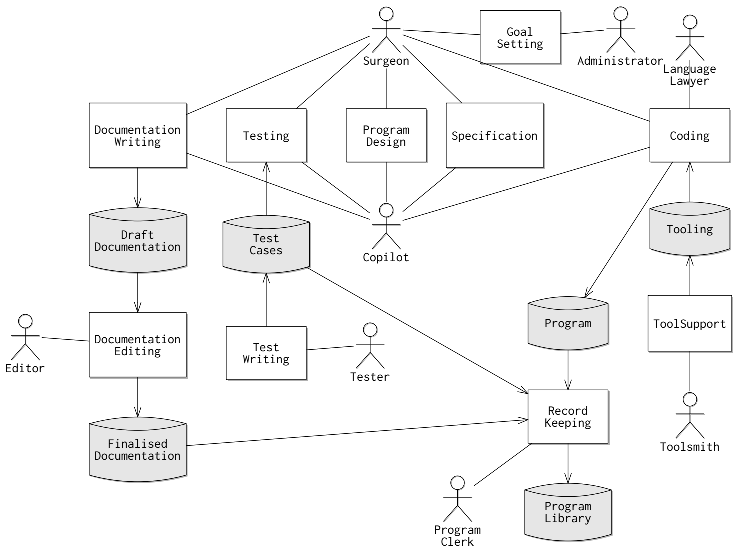


Fig. 1.2. Responsibility Model of the surgical team proposed by Brooks [5]
.

software lifecycle model and we do not attempt to model it as such; its focus is on allocating members of a development team to responsibilities to maximise productivity and minimise organisational overhead.

Inspecting our model reveals a number of issues with the surgical team concept. The relationship between surgeon and the copilot is not well defined, as they share responsibility for almost all functions they perform, excepting administration. The surgical team proposal does not define the process by which the surgeon and copilot divide and allocate tasks, which suggests that this is an additional process that should be implemented when the surgical team is used.

The role of the language lawyer is also shown to be ambiguous. The only concrete responsibility they are involved in is coding, by optimising routines and improving the quality of code where possible. While several other roles (such as acting as an advisor) are suggested they are not formally included.

The record keeping activities of the Program Clerk are shown to be peripheral to the overall team activities. They are not required to perform the main responsibilities of the team and their absence does not prevent any of the core responsibilities from being discharged. We have shown the Program Clerk's responsibility to depend on the production of test cases, documentation and program code, since we *assume* this responsibility includes curation of these artefacts. However, this is not made explicit in the description by Brooks [5] .

Responsibility modelling has highlighted the non-explicit and non-core elements of the system. An absence of assignments or linked responsibilities where expected highlights the parts of the system that are implicit, revealing sections that may not operate in practice as the designers of the system intended. It is also possible to identify non-core actors that are not directly involved with the key responsibilities, and likewise it is clear which actors are involved with the core system functions. Additionally, we can identify how the team will function if lacking certain staff by removing those actors and attempting to meet all the responsibilities. Undischarged responsibilities will be identified, as well as actors that will have increased workload.

1.4.2 Scrum

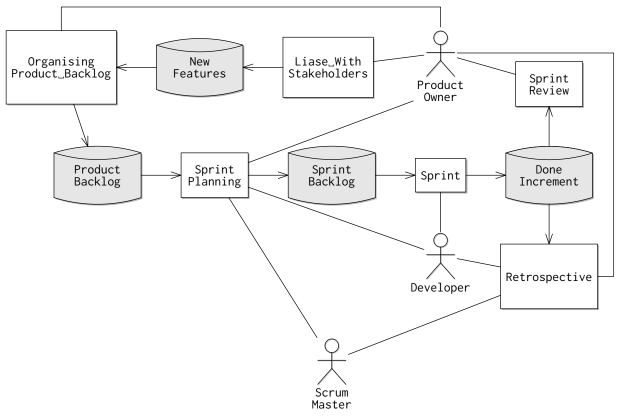


Fig. 1.3. Responsibility Model of Scrum.

The Scrum process is an agile development framework for software development. Scrum is an iterative approach, 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 1.3.

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

In addition, the model suggests that Scrum is a more collaborative process than the Surgical team. Although there are distinct roles for Product Owner and Scrum Master, these do not hold responsibilities that are analogous to the Administrator or Surgeon in the Surgical team. Rather, all roles collaborate in many different responsibilities in different ways. For example, Developers are responsible in collaboration with the Scrum Master and Product Owner for Scrum 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, but with distinct sub-responsibilities.

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.

A responsibility can be decomposed in order to understand how agents collaborate to discharge the overall responsibility. Figure 1.4 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 shares responsibility for coordinating the Daily Scrum. The exact division of responsibilities within the Daily Scrum may be identified by further refinement if desired.

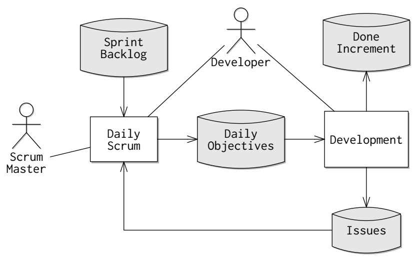


Fig. 1.4. Refinement of 'Sprint' Responsibility.

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. Problems with the new feature are reported as Issues that must be resolved before integration is permitted.

The decomposed model is consistent with the overall model presented in Figure 1.3. 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.

1.4.3 Observations

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 (such as the intra-team behaviour in Scrum) or may reflect uncertainty over the use of that part in practice (such as the language lawyer of the surgical team). 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.

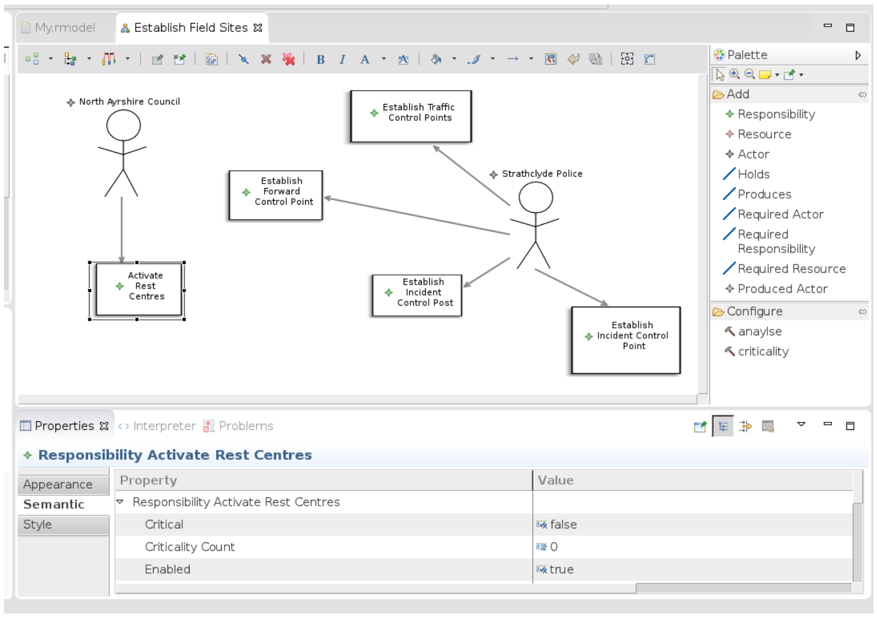
Responsibility modelling is especially effective when comparing deployments or implementations against the theoretical standard. The complexity and variation of social-technical systems means that many real systems vary significantly from their conceptual model, but these differences are not regularly formalised. 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 alleviate this by allowing both high-level and implementation-level models of the system to be produced using the same scheme. The well-defined semantics of responsibility modelling will allow sections of the implementation-level model to be mapped automatically to the relevant responsibilities in the high-level model, so that the completeness of implementations can be checked directly against the original specification.

1.5 Prototype Tool Support

A prototype tool for responsibility modelling has been implemented using Eclipse Sirius, as displayed in Figure 1.5. The tool currently allows the graphical creation and editing of responsibility models, with all types of objects and relations supported. Models are created by dragging components from the toolbar, and once placed in the model they can be rearranged and modified at will. Relations can be added by selecting the appropriate type and selecting the two objects to be related. Model elements can also be enabled and disabled, allowing analysts to examine the effects of individual failures on the wider system. The underlying representation of models is built using the Eclipse Modelling Framework, which provides an additional cross-platform XML representation.

Automatic Analysis The existence of a formal structure for responsibility modelling enables wide ranging analysis to be performed automatically on models produced with this toolkit, with several analysis techniques implemented and more planned.



Responsibility Activate Rest Centres

Property	Value
Responsibility Activate Rest Centres	
Critical	false
Criticality Count	0
Enabled	true

Fig. 1.5. Eclipse-based responsibility modelling tool

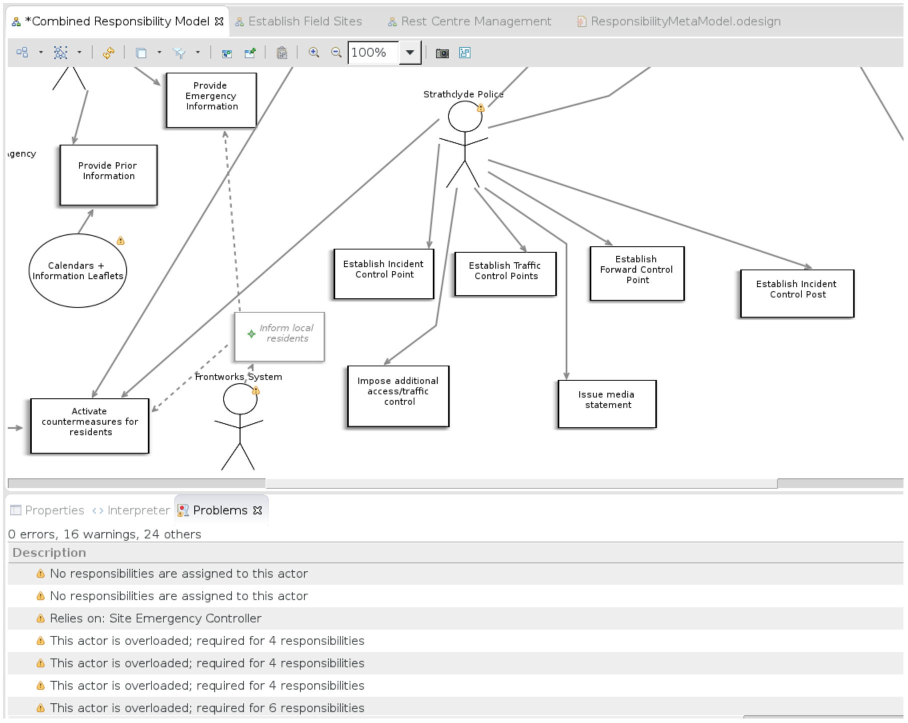
Basic analysis can be performed by triggering a model validation, which performs around half a dozen local checks. Entities are checked for their completeness, locating unproduced resources and actors without assignments. In many cases incomplete elements will indicate that further modelling detail is required, but they may also indicate fundamental shortcomings in the system being modelled.

Thresholds can be set to determine potentially overloaded actors that hold an excessive amount of responsibilities, potentially leading to degraded operation. Actors holding large amounts of responsibilities might become overloaded and perform poorly if attempting to simultaneously manage different tasks. Likewise, resources that are consumed by multiple responsibilities highlight potential issues over resource allocation or exhaustion.

Additionally, for each actor a set of other actors that it relies on is generated - as actors may hold certain responsibilities, but rely on other actors to ensure that they are discharged.

Two specialist forms of analysis are also possible, which operate across the entire model. Criticality analysis detects the most critical entities - the resources, responsibilities and actors that contribute the most to the system and would cause the highest number of responsibility failures if they failed to operate.

The second form of model-wide analysis is responsibility discharge detection, which is augmented with a powerful constraint language. 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


*Combined Responsibility Model
Establish Field Sites
Rest Centre Management
ResponsibilityMetaModel.odesign
100%
Provide Emergency Information
Strathclyde Police
gency
Provide Prior Information
Calendars + Information Leaflets
Establish Incident Control Point
Establish Traffic Control Points
Establish Forward Control Point
Establish Incident Control Post
Inform local residents
Frontworks System
Impose additional access/traffic control
Issue media statement
Activate countermeasures for residents
Properties
Interpreter
Problems

```
0 errors, 16 warnings, 24 others
Description
No responsibilities are assigned to this actor
No responsibilities are assigned to this actor
Relies on: Site Emergency Controller
This actor is overloaded; required for 4 responsibilities
This actor is overloaded; required for 4 responsibilities
This actor is overloaded; required for 4 responsibilities
This actor is overloaded; required for 6 responsibilities
```

Fig. 1.6. Analysis in progress, showing reliance, overload and selectively disabled responsibilities

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 constraints.

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.

In responsibility modelling actors are intended to be roles or positions, rather than actual individuals. However, in any actual implemented systems these roles will clearly be filled by specific actors or organisations. This could lead to vulnerabilities not captured by conventional modelling if an actual individual or organisation takes on the responsibilities of multiple actors. This could lead to overloading and hence a performance risk, or the same individual may hold actor roles that conflict or require them to perform multiple tasks simultaneously. While not currently implemented, we plan to develop tooling that can enable this to be taken into consideration by providing a model view that allows real-world entities to be mapped to actors, enabling automatic detection of overloaded entities and a visualisation of potential conflicts of interest.

1.6 Conclusion

Modelling social-technical systems is often a difficult compromise, with informal methods lacking the rigour needed to enable automatic and semi-automatic analysis while formal methods can struggle to model the complexity of human activity within the system.

Responsibility modelling offers many characteristics that make it a suitable middle ground for modelling social-technical systems, as it combines the flexibility of more informal approaches with a more rigorous structure. In previous work this rigour has not been fully applied, with responsibility modelling used more for structured discussion of a system rather than as a complete model.

In this paper, we have shown that responsibility modelling can be formalised with a fully-defined syntax and consistent semantics. Case studies indicate that it is possible to model non-trivial social-technical systems with formalised responsibility modelling and that useful analysis of these case studies can be carried out by analysing the responsibility model without needing expert knowledge of the problem domain.

Additionally, much of this analysis can be fully automated, or assisted greatly by the use of software tool support. A prototype tool for responsibility modelling and analysis has been developed with initial support for modelling construction and individual analysis, and the well-defined semantics of formalised responsibility modelling allow for a wide range of future tool features, such as automatic completeness, validity and redundancy checking.

References

[1] Basnyat, S., Chozos, N., Johnson, C., Palanque, P.: Incident and accident investigation techniques to inform model-based design of safety-critical interactive systems. In: Gilroy, S.W., Harrison, M.D. (eds.) Interactive Systems, Design, Specification, and Verification, 12th International Workshop DSVIS 2005, July 2005. Lecture Notes in Computer Science, vol. 3941, pp. 51–66. Springer, Newcastle upon Tyne (2006)

[2] Baxter, G., Sommerville, I.: Learning lessons from the failures of socio-technical systems design (April 2008), for submission to Interacting with Computers

[3] Besnard, D., Baxter, G.: Human compensations for undependable systems. Tech. Rep. CS-TR-819, School of Computing Science, Newcastle University, Newcastle upon Tyne, UK (November 2003)

[4] Blyth, A.J., Chudge, J., Dobson, J.E., Strens, M.R.: ORDIT: A new methodology to assist in the process of eliciting and modelling organisational requirements. In: Kaplan, S. (ed.) Proceedings on the Conference on Organisational Computing Systems. pp. 216–227. ACM Press, Milpitas, California, USA (1993)

[5] Brooks, Jr., F.P.: The Mythical Man-Month. Addison Wesley, ninth edn. (1995)

[6] Cholvy, L., Cuppens, F., Saurel, C.: Towars a logical formalization of responsibility. In: Proceedings of the 6th international conference on Artificial intelligence and law. pp. 233–242. ACM Press, Melbourne, Australia (June–July 1997)

[7] Darimont, R., Delor, E., Massonet, P., van Lamsweerde, A.: GRAIL/KAOS: an environment for goal-driven requirements engineering. In: Adrion, W.R. (ed.) ICSE'97: Pulling Together, Proceedings of the 19th International Conference on Software Engineering. pp. 612–613. ACM Press, Boston, Massachusetts, USA (May 1997)

[8] Dewsbury, G., Dobson, J. (eds.): Responsibility and Dependable Systems. Springer-Verlag London Ltd (June 2007)

[9] Dobson, J.E., Sommerville, I.: Roles are responsibility relationships really (October 2005), dIRC Project Technical Report

[10] Feiler, P., Gabriel, R.P., Goodenough, J., Linger, R., Longstaff, T., Kazman, R., Klein, M., Northrop, L., Schmidt, D., Sullivan, K., Wallnau, K.: Ultra-Large-Scale Systems, The Software Challenge of the Future. Software Engineering Institute, Carnegie Mellon University, 4500 Fifth Avenue, Pittsburgh (June 2006)

[11] Feltus, C., Incoul, C., Aubert, J., Gateau, B., Adelsbach, A., Camy, M.: Methodology to align business and IT policies: Use case from an IT policy. In: 2009 International Conference on Availability, Reliability and Security. IEEE Computer Society, Fukuoka, Japan (March 2009)

[12] Feltus, C., Petit, M.: Building a responsibility model including accountability, capability and committment. In: 2009 International Conference on Availability, Reliability and Security. IEEE Computer Society, Fukuoka, Japan (March 2009)

[13] Garion, C., van der Torre, L.: Design by contract deontic design language for multiagent systems. In: Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems. Lecture Notes in Computer Science, vol. 3913, pp. 170–182. Springer Verlag, Utrecht, The Netherlands (July 2006)

[14] Gerber, A., Kotzé, P., van der Merwe, A.: Towards the formalisation of the togaf content metamodel using ontologies. In: Filipe, J., Cordeiro, J. (eds.) ICEIS 2010 - Proceedings of the 12th International Conference on Enterprise Information Systems. vol. 2, pp. 54–64. SciTechPress, Funchal, Madeira, Portugal (June 2010)

[15] Harper, R., Newman, W.: Designing for user acceptance using analysis techniques based on responsibility modelling. In: Tauber, M.J. (ed.) CHI 96

Conference Companion on Human factors in computing systems. pp. 217–218. ACM Press, Vancouver, B.C. (April 1996)

[16] Hause, M.: The sysml modelling language. In: 5th European Systems Engineering Conference. INCOSE, Edinburgh, Scotland, UK (September 2006)

[17] Kobryn, C., Sibbald, C.: Modeling dodaf compliant architectures the telelogic approach for complying with the dod architectural framework. White paper, Telelogic (October 2004)

[18] Lapouchnian, A.: Goal-oriented requirements engineering: An overview of the current research. Depth report, Department of Computer Science, University of Toronto (June 2005)

[19] Leveson, N.G.: A systems-theoretic approach to safety in software-intensive systems. IEEE Transactions on Dependable and Secure Computing 1(1), 66–86 (January 2004)

[20] Leveson, N.G., Dulac, N.: Safety and risk-driven design in complex systems-of-systems. In: Proceedings of the 1st NASA/AIAA Space Exploration Conference: Continuing the Voyage of Discovery. American Institute of Aeronautics and Astronautics, Orlando, Florida, USA (January–February 2005)

[21] Lock, R., Sommerville, I.: Modelling and analysis of socio-technical system of systems. In: 10th International Conference on Engineering Complex Computer Systems. pp. 224–232. IEEE Computer Society, Oxford, UK. (March 2010)

[22] Lock, R., Storer, T., Sommerville, I.: Responsibility modelling for risk analysis. In: Proceedings of European SREL (ESREL) 2009. pp. 1103–1109. Prague, Czech Republic (September 2009)

[23] Meyer, J.J.C., Weiringa, R.J.: Applications of deontic logic in computer science: A concise overview. In: Meyer, J.J.C., Weiringa, R.J. (eds.) Deontic Logic in Computer Science: Normative System Specification, pp. 17–45. John Wiley and Sons Ltd., Chicester, UK. (1993)

[24] Mumford, E.: The story of socio-technical design: reflections on its successe, failures and potential. Information Systems Journal 16, 317–342 (2006)

[25] Padmanabhan, V., Governatori, G., Sadiq, S., Colomb, R., Rotolo, A.: Process modelling: The deontic way. In: Stumptner, M., Hartmann, S., Kiyoki, Y. (eds.) Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling. Conferences in Research and Practice in Information Technology Series, vol. 53, pp. 75–84. ACM Press, Hobart, Australia (January 2006)

[26] Schwaber, K., Beedle, M.: Agile Software Development with SCRUM. Prentice Hall (2001)

[27] Sommerville, I., Lock, R., Storer, T.: Information requirements for enterprise systems. In: Calinescu, R. (ed.) 17$^{\text{th}}$ Monterey Workshop. pp. 266–282. No. 7539 in Lecture Notes in Computer Science, Springer Verlag (March 2012)

[28] Sommerville, I., Lock, R., Storer, T., Dobson, J.: Deriving information requirements from responsibility models. In: Eck, P.V., Gordijn, J., Wieringa, R. (eds.) Advanced Information Systems Engineering, 21st International

Conference, CAiSE 2009. Lecture Notes in Computer Science, vol. 5565, pp. 515–529. Springer Verlag, Amsterdam, Netherlands (June 2009)

[29] Sommerville, I., Storer, T., Lock, R.: Responsibility modelling for civil emergency planning. Risk Management 11(3-4), 179–207 (2009)

[30] Strens, R., Dobson, J.: How responsibility modelling leads to security requirements. In: NSPW '92-93: Proceedings on the 1992-1993 workshop on New security paradigms. pp. 143–149. ACM Press, New York, NY, USA (1993)

[31] Trist, E.: The evolution of socio-technical systems. a conceptual framework and an action research program. Occasional paper 2, Ontario Quality of Working Life Centre (June 1981)

[32] Yu, E.S.: Towards modelling and reasoning support for early-phase requirements engineering. In: 3rd IEEE International Symposium on Requirements Engineering (RE'97). pp. 226–235. IEEE Computer Society (1997)