

# **Design-Time and Run-Time Requirements Modelling for Adaptive Systems**

Kristopher Welsh M.Sc (Dist), B.Sc (Hons)

School of Computing and Communications  
Lancaster University

*A thesis submitted in partial fulfilment of the requirements  
for the degree of Doctor of Philosophy*

September 2010

# Design-Time and Run-Time Requirements Modelling for Adaptive Systems

Kristopher Welsh M.Sc (Dist), B.Sc (Hons)

School of Computing and Communications

*A thesis submitted in partial fulfilment of the requirements for  
the degree of Doctor of Philosophy*

September 2010

This thesis explores the construction, enrichment and use of requirements models for adaptive systems. This thesis proposes the enrichment of adaptive systems' requirements models with additional tracing information, preserving the rationale behind configuration decisions. The preserved rationale can be utilised at design time, allowing decisions to be re-taken in the context of new or changed information and allowing for the identification of areas of uncertainty in understanding. The preserved rationale can also be used by a system itself, at run time, allowing it to adapt its behaviour to contexts not fully envisaged at design time.

This thesis presents the ReAssuRE modelling process, which defines modelling perspectives from which some classes of adaptive system may be viewed. ReAssuRE models embed the described tracing information, and may be interpreted and reasoned with by a suitably constructed system at run time. This thesis presents tool support, allowing adaptive behaviour to be derived directly from ReAssuRE models. Finally, this thesis presents proof of concept components that allow a system to reason with ReAssuRE models, transforming them in response to monitored data, and derive new adaptive behaviour.

# Declaration

I declare that this thesis is my own work, and that the work presented in this thesis is work completed by myself. The work presented in this thesis has not been presented in substantially the same form for a higher degree elsewhere. Work that has previously been published was completed as part of the doctoral program, and has been clearly identified in all cases.

# Acknowledgements

This thesis represents the culmination of four years' work, and the end of perhaps the most challenging and enjoyable chapter of my life thus far. My most sincere thanks go to my supervisor, Pete Sawyer, whose flawless instinct governing when to let work take its course and when to reign in, has made possible an achievement that for the longest time I thought impossible.

In kijita, “Omwana ni wa bhone” means that a child's upbringing belongs to the community, regardless of biological parentage. If true, then a researcher's development may be owed to an entire department. Those in the department such as Nelly Bencomo, Danny Hughes, Gerald Kotonya, Andre Oboler, Benjamin Green and Jane Holt have shaped my work and development in countless ways, and to them I am forever grateful.

# Table of Contents

<b>1 Introduction.....</b>	<b>1</b>
1.1 Overview.....	1
1.2 Motivation for Research.....	3
1.3 Research Objectives.....	5
1.4 Research Method.....	8
1.5 Novel Contributions.....	9
1.6 Scope of the Thesis.....	10
1.7 Structure of the Thesis.....	11
<b>2 Background.....</b>	<b>13</b>
2.1 Autonomous Systems and DASs.....	13
2.2 Domains Promoting Increasing Autonomy.....	16
2.3 Existing Systems.....	18
2.4 Existing Adaptation Infrastructures.....	23
2.4.1 Multi-Agent Adaptive Infrastructures.....	23
2.4.2 Architectural Adaptive Infrastructures.....	26
2.4.3 AI Planning for Adaptation.....	28
2.4.4 Explicit Programming Language Adaptation Support.....	29
2.5 Chapter Conclusion.....	30
<b>3 DAS Requirements Modelling &amp; Monitoring.....</b>	<b>32</b>
3.1 Requirements Modelling.....	33
3.1.1 KAOS-based DAS Modelling.....	36
3.1.2 i*-based DAS Modelling.....	38
3.2 Requirements Monitoring.....	41
3.3 Chapter Conclusion.....	44
<b>4 ReAssuRE Modelling for DASs.....</b>	<b>47</b>
4.1 The LoREM Process.....	47
4.2 Level One Modelling.....	49
4.2.1 Recording Rationale with Claims.....	52

4.2.2 Examining Assumptions with Claim Refinement Models.....	56
4.2.3 Benefits and Limitations.....	58
4.3 Level Two Modelling.....	65
4.4 Chapter Conclusion.....	71
<b>5 ReAssuRE at Design Time.....</b>	<b>72</b>
5.1 Traceability in ReAssuRE models.....	72
5.1.1 Change in DAS Requirements.....	73
5.1.2 Tracing Examples.....	77
5.2 Policy Derivation.....	96
5.3 Requirements Validation.....	105
5.4 Chapter Conclusion.....	109
<b>6 ReAssuRE at Run Time.....</b>	<b>111</b>
6.1 Run-Time Representation of ReAssuRE models.....	112
6.2 Assumption Monitoring and “broken” Label Propagation.....	113
6.3 ReAssuRE m-DAS architecture.....	121
6.4 Uncovering Emergent Behaviour in m-DASs.....	126
6.5 Chapter Conclusion.....	133
<b>7 Evaluation.....</b>	<b>135</b>
7.1 ReAssuRE models for the GridStix System.....	138
7.1.1 Level-One Strategic Rationale Models.....	140
7.1.2 Level-One Claim Refinement Models.....	147
7.1.3 Level Two Models.....	151
7.2 Tracing Examples.....	152
7.3 Policy Derivation.....	161
7.4 Policy Simulation.....	164
7.5 Requirements Validation.....	167
7.6 Model-Driven Adaptation.....	175
7.7 Testing for Emergent Behaviour.....	182
7.8 Chapter Conclusion.....	188
<b>8 Conclusions and Future Work.....</b>	<b>190</b>

8.1 Research Questions Revisited.....	190
8.2 Research Hypothesis Revisited.....	193
8.3 Limitations.....	195
8.4 Future Work.....	197
8.5 Closing Remarks.....	199
<b>Appendix A – Image Viewer's Adaptation Policy.....</b>	<b>200</b>
<b>Appendix B – GridStix Adaptation Policy.....</b>	<b>201</b>
<b>Appendix C – Annotated GridStix Claim Refinement Models.....</b>	<b>206</b>
<b>Appendix D – Pseudocode for Policy Generation.....</b>	<b>210</b>
<b>References.....</b>	<b>211</b>

# Table of Figures

Figure 1: Kramer and Magee's Adaptive Architecture [3].....	26
Figure 2: Level 1 LoREM SD Model for Adaptive Image Viewer.....	50
Figure 3: Level 1 LoREM SR Model for Adaptive Image Viewer's S2 Target System.....	50
Figure 4: ReAssuRE Model of Image Viewer's S2 Target System.....	54
Figure 5: Claim using a “break” Contribution Link.....	55
Figure 6: Claim Refinement Model for Adaptive Image Viewer's S2 Target System.....	57
Figure 7: Partial $i^*$ SR Model for Video Streaming System.....	59
Figure 8: Claim-Augmented $i^*$ SR Model of Video Streaming System.....	60
Figure 9: Claim Refinement Model for Codec & Bitrate for Video Streaming System.....	61
Figure 10: LoREM Level Two Model of Adaptive Image Viewer's S1-S2 Transition.....	66
Figure 11: ReAssuRE Level Two Model of Image Viewer's S1-S2 Transition.....	68
Figure 12: ReAssuRE Level Two Model of Image Viewer's S2-S1 Transition.....	70
Figure 13: Transferred Elements of Adaptive Image Viewer's S3 Model.....	79
Figure 14: Final S3 Level One Model for Adaptive Image Viewer.....	80
Figure 15: Claim Refinement Model for Adaptive Image Viewer's S3 Target System.....	81
Figure 16: “Broken” Label Propagation in Claim Refinement Model.....	83
Figure 17: Adaptive Image Viewer's S2 Target System with Broken Assumption.....	83
Figure 18: Adaptive Image Viewer's S2 Target System with New Claim.....	84
Figure 19: Adaptive Image Viewer's S2 Target System with New Cache Available.....	86
Figure 20: Claim Refinement Model with Rationale for Learning Cache in S2.....	87
Figure 21: Complete Claim Refinement Model Supporting Learning Cache's Use.....	88
Figure 22: Adaptive Image Viewer's S2 Target System with Learning Cache Rationale.....	89
Figure 23: Simplified SR Model Showing Learning Cache's Selection in S2.....	90
Figure 24: Simplified Claim Refinement Model Supporting Learning Cache's Use.....	90
Figure 25: Adaptive Image Viewer's S2 Target System with Cost Contributions.....	91
Figure 26: ReAssuRE Model of Adaptive Image Viewer's S1 Target System.....	94
Figure 27: Adaptive Image Viewer's S1 Target System after Requirement Change.....	95
Figure 28: ReAssuRE Models of Adaptive Image Viewer's Two Target Systems.....	97



Figure 29: Snippet of Adaptive Image Viewer's Adaptation Policy.....	98
Figure 30: Screen-Shot of Policy Generation Tool.....	101
Figure 31: Screen-Shot of the Genie DSL Derivation Tool.....	102
Figure 32: Genie model of Adaptive Image Viewer's Adaptive Behaviour.....	103
Figure 33: Uncertainty Propagation in S2 Claim Refinement Model.....	107
Figure 34: Modified S2 Claim Refinement Model.....	117
Figure 35: Level 1 SR Model of Image Viewer S2 with Alternate Claim.....	118
Figure 36: Claim Refinement Model for Image Viewer S1 With Alternate Claim.....	118
Figure 37: Level 1 SR Model of Image Viewer S2 after Claim Inversion.....	120
Figure 38: Level 1 Model of Image Viewer S2 after Removal Modification.....	121
Figure 39: m-DAS Architecture Model.....	122
Figure 40: Single Monitoring Rule from a Monitoring Policy.....	123
Figure 41: Implemented Model Transformer Architecture.....	125
Figure 42: Annotated Claim Refinement Model for Learning Cache.....	130
Figure 43: Level-One Strategic Dependency Model for GridStix System.....	139
Figure 44: Level 1 SR Model of GridStix S1 Target System.....	141
Figure 45: Level 1 SR Model of GridStix S2 Target System.....	144
Figure 46: Level-One SR model for GridStix S3 Target System.....	146
Figure 47: Claim Refinement Model for GridStix S1 Target System.....	147
Figure 48: Claim Refinement Model for GridStix S2 Target System.....	148
Figure 49: Claim Refinement Model for GridStix S3 Target System.....	149
Figure 50: Level 2 Model of GridStix S1 to S2 Transition.....	151
Figure 51: Modified GridStix S3 Claim Refinement Model: Broken Assumption.....	153
Figure 52: Modified SR Model for GridStix S3 Target System: Broken Assumption.....	154
Figure 53: Modified SR Model For GridStix S3 Target System: New Rationale.....	155
Figure 54: Modified Claim Refinement Model for GridStix S3: New Rationale.....	155
Figure 55: Level 1 SR Model for new GridStix S4 Target System.....	157
Figure 56: Claim Refinement Model for new GridStix S4 Target System.....	158
Figure 57: Level Two Model for GridStix S1 to S4 Transition.....	159
Figure 58: Screen-Shot Genie DSL Model Derivation Tool.....	161
Figure 59: GENIE Model of GridStix DAS Derived from ReAssuRE Models.....	162

Figure 60: Tool Generating Adaptation Policies from ReAssuRE Models.....	163
Figure 61: Snippet of Generated XML Adaptation Policy.....	163
Figure 62: GridStix Simulator Display.....	166
Figure 63: GridStix S3 Claim Refinement Model with Claim Classification.....	170
Figure 64: Snippet of GridStix Monitoring Policy.....	177
Figure 65: Modified GridStix S3 Claim Refinement Model.....	178
Figure 66: Modified GridStix S3 Strategic Rationale Model.....	179
Figure 67: More Extensively Modified GridStix S3 Claim Refinement Model.....	180
Figure 68: More Extensively Modified GridStix S3 Strategic Rationale Model.....	181
Figure 69: GridStix S3 Claim Refinement Model with Testing Claim Classification.....	185

## Table of Tables

Table 1: Claim Contribution and Attached Link Inclusiveness.....	55
Table 2: Support for Traceability Requirements of Identified Change Classes.....	77
Table 3: Command-Line Arguments of Policy Generation Tool.....	100
Table 4: Testing Scenarios Required to Cover the Categories of Claims.....	131
Table 5: Claim Combinations for GridStix S3 Validation Scenarios.....	169
Table 6: GridStix Validation Scenarios by Target System.....	170
Table 7: GridStix Validation Scenario Reduction using Claim Classification.....	172
Table 8: GridStix Test Case Reduction using Claim Classification.....	186

# 1 Introduction

---

## 1.1 Overview

Today's software is increasingly complex, consisting of ever larger numbers of distinct, interdependent components. Furthermore, this software is increasingly expected to operate in ever more changeable environments, which necessitates software of even greater complexity. Autonomous systems that monitor, configure, tune and heal themselves independently of human interaction are perceived as necessary to allow systems to operate in volatile environments, and to mitigate this increasing complexity [1]. Although, for most tasks, completely autonomous systems are beyond our reach today, we are creating systems that possess some ability to alter their behaviour as the operating environment changes [2].

Dynamically Adaptive Systems (DASs), sometimes referred to as self-managed systems [3], monitor their operating environment at run time, and adjust their behaviour to better achieve system goals in response to detected changes. Many current DASs offer only a finite set of potential configurations, and the only real autonomy is found in the system's ability to select an appropriate configuration for the current environmental context.

Despite the potential for DASs to mitigate increasing software complexity, adaptive behaviour is in itself somewhat complex. A state of the art DAS utilises a reusable adaptive infrastructure to enable adaptation [2]. An adaptive infrastructure codifies per-application adaptive behaviour in adaptation policies, which explicitly state which configurations are to be used under which environmental contexts.

The rigidity in pre-specifying configurations and tying them to specific contexts introduces a limitation in a DAS's adaptive capability: only environmental volatility anticipated at design time can be addressed. To allow a DAS to tolerate completely

unexpected environmental change, a greater degree of autonomy is required. There are a number of ways in which a DAS may be able to operate with a greater degree of autonomy: it could use some form of artificial intelligence (AI), for example. However, merely increasing the level of intelligence underpinning configuration selection decisions does not remove the limitation: the DAS is still only cycling through a number of pre-set, human-specified configurations. To create a system that is genuinely more autonomous, it is necessary for the DAS to devise and adopt configurations that were not pre-specified by humans when it is faced with operating conditions outside those envisaged at design time.

There is an inherent danger in granting systems an ever increasing degree of autonomy: that we lose the ability to predict their behaviour and that we can no longer assure that the autonomously devised configuration is optimal, or even correct. How can we expect to deliver assurance if we aren't even sure what the system will do? There appears to be a fundamental trade-off between autonomy and predictability [4].

DASs present challenges to the software engineering research community in terms of their potential scale and complexity. Software engineering is optimised for the design and specification of systems that operate in steady, non-volatile environments, and for verifying the behaviour of a constructed system to ensure that it really does behave as expected. However, for a DAS these tasks have to be performed for each available configuration, and each environmental context. For a DAS, the software engineering workload can, potentially, be orders of magnitude greater than for a non-adaptive, so-called static system.

This thesis presents the ReAssuRE modelling approach, which allows the behaviour of a DAS to be modelled during the requirements engineering process. The models feature enhanced traceability, which improves the changeability of the models. The potentially poorly understood, volatile, nature of the operating environments in which DASs are deployed, and the difficulty of predicting the behaviour of a DAS in these challenging environments promotes changeability as a key concern for DAS requirements models.

This thesis also introduces a new class of DAS, called a model-driven Dynamically Adaptive System (m-DAS). This class of DAS uses design-time models to guide run-time adaptation decisions, and is a natural application of the emerging models@run.time paradigm [5] to DASs. This thesis demonstrates how ReAssuRE models can be used by a m-DAS to devise new configurations in response to unforeseen environmental conditions by combining components in combinations not prescribed at design time, whilst operating in the same way as a DAS in the conditions for which the system was designed.

To address the autonomy vs. predictability trade-off, this thesis also presents a model-directed testing approach, that uses model analysis of the ReAssuRE models constructed for a m-DAS to identify scenarios that offer the potential to prompt a m-DAS to display emergent behaviour. A means of analysing the likelihood of the scenarios occurring with a view to directing limited testing resources to the most likely of these key scenarios is also presented.

## 1.2 Motivation for Research

Systems able to configure themselves during installation, optimise their behaviour during day-to-day use, heal themselves in the event of problems and protect themselves from attack are a staple of science-fiction writers, and represent a utopian view of computing shared by many [1] [6]. Today's reality is, however, somewhat different. Systems are typically configured by hand in a process prone to error, with optimisation occurring more frequently by means of maintenance performed whilst the system is offline than in real-time by the system itself. System protection tends to come in the form of designers anticipating specific threats and taking steps to mitigate them, although some systems can adopt special modes designed to allow the system to continue operating when attacked [7] We are also seeing systems that possess some ability to recover from error, or at least possess a reduced functionality “limp” mode, activated indiscriminately in the event of any serious error [8].

Although the degree of autonomy described in the above scenarios is low, the commonality among all of these lightly autonomous systems is the presence in each of an alternate mode of operation, that the system adopts when appropriate. As such, it becomes trivial to envisage a system with more numerous modes of operation, that can tolerate a wider range of environmental variance. A DAS is a useful abstraction over this class of system, with the term DAS imposing no restriction in the number, type or purpose of the modes a system may adopt, or the frequency with which they are adopted. A more advanced DAS offers a greater number of modes, tailored to a greater range of variability in the operating environment, with greater intelligence driving the selection decisions. Essentially, a DAS embodies the state-of-the-art in autonomic computing.

DASs are difficult to design and specify because each mode of operation in a DAS has all of the complexity of a complete system, essentially multiplying the associated software engineering workload. Furthermore, if the operating environment itself is volatile, this volatility needs to be factored into the requirements engineering process, and there's the issue of how the system is to decide which mode of operation should be used under each environmental context. Combined, these factors increase the complexity of the requirements engineering process and the research community is beginning to recognise DASs as a class of system that require special treatment during the RE process [9] [10].

In much the same way as design and specification workload increases with the number of modes of operation a DAS supports, the testing workload likewise increases dramatically. Not only does each mode of operation need testing thoroughly, but also the mechanism by which the need for adaptation is identified, the mechanism responsible for selecting appropriate configurations, and the mechanism effecting adaptation need testing.

For a DAS designed as an aggregation of individual components, its configurations are essentially different combinations of components selected from a fixed set. In such a DAS, it may be possible to offer some degree of assurance by testing the individual components, rather than complete configurations. This testing method offers only partial coverage, and clearly impacts on the level of assurance it is possible to offer. Furthermore, this testing method would only reduce workload in DASs in which the number of modes of operation

is high and the number of distinct components is low. A DAS using other techniques may require every mode of operation to be tested in much the same way as a complete static, non-adaptive, system needs to be tested, thus creating an overwhelming testing burden.

The research is, therefore, motivated by three key concerns:

1. The difficulty in performing RE for a DAS
2. The difficulty in testing a DAS
3. A deficiency in the ability of a DAS to adapt to environmental contexts not envisaged, understood and designed for prior to deployment.

## 1.3 Research Objectives

At its highest level, the research aims to improve the Software Engineering support available to autonomous system developers. More specifically, the research aims to improve modelling support for DASs during the Requirements Engineering process, and to maximise the usefulness of the generated models later in the software engineering process, potentially even at run time.

The later use of requirements models that is targeted by this thesis is in controlling the adaptive behaviour of a DAS. The adaptive behaviour of a DAS is that concerned with the configuration changes made in response to changes in the environment. This thesis explores whether this adaptive behaviour is specified solely from configuration decisions taken during the RE process, and whether enough information about the decisions can be codified in a requirements model to allow the model to be used as a specification of a DAS's adaptive behaviour.

If requirements models are to be used in this way, the purpose of the models will be transformed from illustrative and communicative to prescriptive. Thus, the issues of accuracy and currency become crucial: an erroneous or out of date model will yield



improper behaviour from the deployed DAS. The ability to re-visit a model after a requirements change, for example, allowing the analyst to establish whether changes are necessary and make them, becomes essential.

The use of a collection of requirements models at run time, to guide a DAS's adaptation, would involve the DAS itself analysing the models, adjusting them in response to environmental changes, and using the changed models to prescribe new adaptive behaviour. As discussed in Section 1.1, this additional autonomy comes at a price of predictability. Delivering assurance in a relatively unpredictable system operating in a volatile environment is challenging. This thesis explores how the degree of assurance that can be offered with incomplete testing coverage can be maximised.

As such, the research aims to answer the following questions:

1. To what extent is a DAS's adaptive behaviour merely a derivation of environmental analysis and configuration decisions?
2. Can the information from the environmental analysis and configuration decisions be codified in models, and is it useful to do so?
3. Given that both the information from the environmental analysis and the configuration decisions are subject to change, how can the workload of deriving a DAS's adaptive behaviour be reduced?
4. How can a system be designed with a greater degree of autonomy than current state-of-the-art DASs, and is the extra autonomy useful?
5. How can the testing workload be managed in systems with greater autonomy?

The extent to which a DAS's adaptive behaviour is a derivation of environmental analysis and configuration decisions is a key issue. If entirely, it becomes possible to reduce the workload in prescribing the adaptive behaviour explicitly using a specification document by using a model instead. The workload associated with actually creating the

DAS's adaptive behaviour could even be eliminated if it is possible to perform the derivation automatically.

If the adaptive behaviour of a DAS can be derived from the results of environmental analysis and configuration decisions, then it would be desirable to capture both pieces of information in some kind of model. Deriving adaptive behaviour from the model should be both efficient and repeatable, and would allow the DAS's behaviour to be altered in response to changes in the model or the underlying understanding. The second research question is concerned with this adaptive behaviour modelling.

The third research question is concerned with automating the derivation of a DAS's adaptive behaviour directly from the models mentioned in the second research question. Automating the derivation of adaptive behaviour would not only reduce workload associated with the task, but also offer the possibility of allowing a DAS to perform the derivation autonomously, in response to changes in its own models.

The fourth research question stems from the suggestion made in the previous paragraph. If a DAS can change its own model in response to deficiencies in the model or changes in the environment, and can derive its own adaptive behaviour from the changed model, this new system would offer a greater degree of autonomy than current DASs. The question of whether this extra autonomy could prove useful will require analysis, and the question of whether the extra complexity introduced by the autonomy will justify any extra usefulness will remain.

The fifth and final research question concerns the testing of a system with greater autonomy. The testing of a DAS is already vastly more challenging than for a non-adaptive, static system in terms of scale. Increasing a system's autonomy further will likely compound this problem, and may place the complete testing of such a system beyond the realms of feasibility for all but the simplest systems, and all but the biggest budgets.

## 1.4 Research Method

Most research in the broad field of Software Engineering can be characterised as either qualitative or quantitative, although it is common for research to exhibit features of both paradigms [11]. Quantitative research focusses on the collection and analysis of measured numerical data, with underlying trends identified by deduction and compared to a testable hypothesis, disproving or supporting it. However, there are many research topics within Software Engineering which provide little or no opportunity for quantitative analysis, or for which quantitative analysis of available numerical data would offer limited insight. Research practice in these areas typically follows a qualitative approach, which is more subjective, involving the interpretation of data to refute or support a theory. As such, there is no single well-established research approach in the field of Software Engineering [12] or more specifically Requirements Engineering.

This thesis's work lies within the modelling methods area of RE, which is more of a synthetic discipline than analytic. Synthetic disciplines are concerned with invention and improvement, whereas analytic disciplines are concerned with discovery [13]. The modelling methods research area concerns itself with the invention and improvement of mechanisms for representing systems through part or whole of the RE process, or indeed later into the Software Engineering process. The hypothesis typically under test in such research is that the invention or improvement will prove useful in some specific context. In this light, this thesis's hypothesis is:

When performing early-phase RE for a Dynamically Adaptive System, the recording of additional tracing information will better support change later in the software engineering process. Recorded tracing information can be used during development to derive the adaptive behaviour of a DAS (the concern of switching from one configuration to another as the environment changes), and by the DAS itself after deployment to better adapt to unforeseen conditions.

This thesis argues that the volatile and uncertain nature of DASs' operating environments promotes the importance of an effective and efficient change management process, and offers increased potential for a deployed DAS to encounter conditions that were not envisaged *a priori*.

To test this hypothesis, this thesis uses two case studies. The first, is an illustrative case study to demonstrate the method and its potential benefits, providing proof of concept. This case study was introduced in [14] to provide proof of concept of the ability of DASs to tolerate contextual variability, so it is fitting that methodological support for such systems is depicted in the same terms. The second case study assesses the method's relevance and scalability in a larger, real-world DAS. There are only a limited number of DAS exemplar systems, and a desire to validate the method's benefits on a real-world system colours the choice of second case study.

The use of case studies to validate work that by nature is both synthetic and qualitative is discussed by Perry *et. al* [15], who argue that case studies are well suited to methodological research. They argue that case studies are well suited to answering questions of *how* and *why* a method works, is used, or can be improved in real-world settings in which there is little experimental control.

## 1.5 Novel Contributions

This thesis presents the ReAssuRE modelling process, which features enhanced traceability; promoting ease-of-change throughout the RE process and beyond. The models are also amenable to automated analysis.

Demonstrating the ReAssuRE models' amenability to automated analysis, this thesis presents the TelosTools model parser, which allows existing ReAssuRE models to be formalised, loaded, understood and reasoned with either by a DAS itself at run time or by specialised tools, which offer the possibility of developing tool support for checking, using and reasoning about ReAssuRE models. One such tool, designed to support the hypothesis

introduced in Section 1.4, generates so-called adaptation *policies* automatically from a set of ReAssuRE models. Adaptation policies are used by some adaptive infrastructures to codify a DAS's adaptive behaviour and are discussed further in Section 2.4.1.

Although DASs are a significant class of system in terms of autonomous systems research, the level of autonomy offered is currently relatively low, as highlighted by the third motivating concern presented in Section 1.2. Addressing this deficiency, and utilising the ability to derive adaptive behaviour from ReAssuRE models discussed previously, this thesis introduces a new class of autonomous system. A model-driven DAS, or m-DAS, uses design-time models such as ReAssuRE models to guide its run-time adaptation. This thesis demonstrates that this new class of system possesses some limited ability to redress deficiencies or errors in the design-time models, modify the models, and derive new adaptive behaviour from the modified models.

This thesis presents a model-directed testing method by which scenarios that do not offer the potential for a m-DAS to derive new adaptive behaviour, and thus offer no potential to display emergent behaviour, can be discarded. The remaining scenarios can be categorised by likelihood in order to allow limited testing resources to be directed at the most likely of the scenarios that could potentially uncover emergent behaviour. This pragmatic approach would allow some level of assurance to be afforded an autonomous system through testing, without requiring the impossible individual testing of every possible combination of environmental factors and configurations.

## 1.6 Scope of the Thesis

This thesis focusses primarily on requirements modelling for dynamically adaptive systems. Although this thesis seeks to make a contribution to the requirements engineering of autonomous systems as a whole, the modelling approach presented is designed specifically for DASs. This thesis discusses requirements engineering for systems

with greater autonomy, and presents one such class of system. However, the class of system presented are an evolution of DASs, rather than a revolutionary new system.

Also, requirements traceability for, and the testing of DASs, are discussed at length, but this discussion is constrained to the potential benefit derived from the early-phase requirements modelling approach. The modelling approach itself may also be applicable to non-adaptive systems, still delivering benefit in terms of traceability which may be valuable in some classes of system. However, no work has been undertaken to assess the approach in non-adaptive systems, and no claims of suitability are made.

This thesis concerns itself solely with environments that can be partitioned cleanly into distinct contexts, enabling a specific configuration to be tailored to a defined set of environmental circumstances. This thesis makes no claim that all problem domains for which a DAS could be conceived can be partitioned as such, and this focus introduces a key limitation to the work. There are approaches to designing, specifying and delivering autonomous systems without this limitation, but work is at an early stage, with reference to it made purely to provide context.

It should be noted that even an environment well suited to partitioning is likely to still have a relatively high degree of uncertainty in its nature. The act of partitioning further increases the level of uncertainty with which the DAS has to operate, given the possibility of the partitioning itself being sub-optimal. Although a DAS with greater autonomy may be able to overcome some partitioning deficiencies, this potential isn't explored fully and will be left to future work.

## 1.7 Structure of the Thesis

Chapter 2 of the thesis presents a literature review of the autonomous systems and DAS research area as a whole, examining the scenarios for which such systems are conceived, how they are constructed and tested, and detailing the systems already in existence.

Chapter 3 presents a more focussed literature review, examining in detail the efforts of the RE research community to support DAS specification with modelling techniques. The LoREM modelling approach [16], on which much of the research is based, is detailed in particular.

Chapter 4 presents the ReAssuRE process, detailing the extensions made to the existing LoREM modelling approach, how ReAssuRE models are constructed and the information they are designed to capture. The chapter also demonstrates how notation style affects the precision, richness and complexity of the completed model.

Chapter 5 presents part of the thesis's first case study, focussing on the use of ReAssuRE models at design time. The chapter demonstrates how the additional tracing information codified in ReAssuRE models can be used to manage change, and to identify areas of uncertainty in system or environmental understanding. The chapter also details how adaptive behaviour can be derived from ReAssuRE models automatically.

Chapter 6 presents the second part of the proof-of-concept case study, continuing from Chapter 5 and focussing on the use of ReAssuRE models at run time. The chapter demonstrates the construction of a m-DAS capable of loading and reasoning with ReAssuRE models to guide adaptation. Finally, the chapter details the model-directed testing method by which a degree of assurance can be afforded to a m-DAS when testing resources are too limited for complete testing.

Chapter 7 presents a larger, more complex case study in which the ReAssuRE process has been applied to a real-world DAS to assess the effectiveness of the ReAssuRE Process, with respect to the objectives of the research.

Chapter 8 concludes the thesis, drawing discussion to a close. The chapter includes a brief summary of the thesis, and revisits the research questions, objectives and hypothesis introduced in Sections 1.3 and 1.4. The chapter includes a discussion of the work's limitations and potential future enhancements.

## 2 Background

---

This chapter presents a relatively broad survey of the autonomous systems research area as a whole, to illustrate how DASs represent an important step towards fully autonomous systems. As the chapter progresses, the focus shifts towards existing systems that display some degree of autonomy, and the technology behind and examples of state-of-the-art DASs. The state of the art with regard to software engineering support for DASs is examined in the next chapter.

DASs are beneficial when dealing with volatile, uncertain environments. An ability to adjust its own behaviour allows a DAS to operate effectively in a broader set of conditions than a non-adaptive system, which means that a DAS can be created to operate in environments for which it was previously thought of as infeasible to do so. As a result, DASs are typically expected to operate in environments that are complex and changeable, and often incompletely or misunderstood. Relatively few additional environments are made feasible by considering a DAS, and the next section discusses the type of environment for which a DAS is suitable. Also discussed are the broad types of adaptation a DAS may utilise, and the types of environment each is more suited to.

### 2.1 Autonomous Systems and DASs

A system that is self-managing, self-configuring, self-tuning, self-repairing and self-maintaining is a staple of science-fiction [10]; and the potential of such a fully autonomous system is easy to see, with a particularly striking example presented by Oreizy *et. al.* [6]. These systems are, however, some distance away from becoming reality. At present, we are starting to create systems that possess a limited ability to perform self-configuration as a means to achieve a degree of autonomy. These relatively primitive autonomous systems are termed self-adaptive, or dynamically adaptive systems.



McKinley et. al. [17] identified two distinct types of adaptive behaviour a system can exhibit. The first, named “parameter adaptation” involves the modification of some internal parameter or flag that causes the system to exhibit a change in behaviour. Such adaptation is ideally suited for switching between a small number of pre-defined behaviours according to fixed criteria. So-called context-aware systems [18] often utilise this form of adaptation, which can be viewed as the more rudimentary form of adaptation.

Parameter adaptation's greatest weakness is that the system's behaviour must be defined in its entirety at compile-time [17], along with the logic that determines the switching between different behaviours, or more accurately the setting of the parameter. As such, any change in the environment that was not envisaged at design time cannot be adapted to without taking the entire system off line for code or components to deal with the new environmental situation to be added.

A proposed adaptive system that is intended for operation in an environment that has relatively few, but well understood variances, can take advantage of parameter adaptation to minimise additional complexity. It could be argued that the only time that parameter adaptation proves insufficient is when a system encounters an unforeseen context. Although the system possesses some ability to adjust its behaviour, it remains relatively “brittle” and has no means to adapt to the unexpected conditions. Unfortunately, this lack of malleability could well cause the system to fail, and if dependability is an issue for the proposed system, parameter adaptation could prove unsuitable. Thus, the use of parametric adaptation depends on full faith being held in the environmental analysis. In practice, as environmental complexity and uncertainty increases, parameter adaptation scales badly, and other approaches are favoured.

More advanced forms of parameter adaptation can utilise artificial intelligence (AI) to switch between available behaviours. Such approaches use machine learning techniques to record effective combinations of parameters in certain situations for later re-use, and reason with this knowledge to allow systems to configure themselves when encountering previously unseen conditions.

Parameter adaptation encompasses so-called moded systems, which have been created for some time. These systems offer several differing modes of operation, selected either manually or automatically to allow some degree of flexibility in the system's operation. For example, it is common for automotive engine management systems to possess a “limp home” mode, which allows the engine to run with reduced power with the system in a simple configuration in the event of serious error, allowing the vehicle to be driven home before requiring repair. Parameter adaptation is, as such, not a new concept; its importance being to contrast the second, emerging type of adaptation: compositional adaptation.

The second type of adaptation is known as “compositional adaptation” or sometimes “structural adaptation”. Compositional adaptation refers to the fact that the structural elements that make up such a system can be swapped, rearranged or removed dynamically in response to changes in context. A system utilising compositional adaptation can use different combinations of these structural elements to radically alter its behaviour. Simpler compositionally adaptive systems may still define the combinations of structural elements that are to be used in which contexts explicitly and rigidly by hard-coding strategies. More advanced systems, however, allow the strategies to be updated or adjusted at run time supporting tuning and maintenance. More advanced still, it may be possible for a system to develop the strategies itself, perhaps in response to some unforeseen or previously unseen environmental conditions. There is a clear path by which systems using compositional adaptation can operate with increasing autonomy.

State-of-the-art DASs rely on the specification of which structural elements are to be combined in what manner under which contexts. The software, and more specifically requirements engineering support for this process forms the focal point of much of this thesis, and is discussed further in Section 3.1. The step forward (in terms of autonomy) to systems that can develop or revise these specifications is another central theme, and is discussed at length in Chapter 6.

A DAS's ability to tailor its behaviour to accommodate changes in context is analogous to the system performing tuning or maintenance autonomously. However, a DAS's ability

to tune itself to *some* conditions, or to perform *some* maintenance online does not remove the need to perform the tasks in circumstances outside those for which it was specifically designed. In this respect, a DAS may have a wider operating envelope than a traditional, static system, but there are still many potential sources of change that may necessitate human intervention. Ideally, as system autonomy increases, so too should the system's operating envelope, and the need for human intervention should decrease.

Software Engineering has been slow to support DASs explicitly. Regardless, there are several domains in which we are seeing DASs created currently, and these are discussed in the next section.

## 2.2 Domains Promoting Increasing Autonomy

The desire to operate in certain domains acts to drive the development of DASs. In these domains, the expected operating environment exhibits significant volatility, and systems are being developed with adaptive capability to allow them to function in the face of uncertainty. This section discusses some of the more prevalent examples of domains in which DASs have proven beneficial.

Perhaps the best known domain in which autonomy is becoming prevalent is that of the network-centric system. In this context, software is increasingly expected to operate on lossy, congested and sometimes unreliable wireless networks [19] [20]. In some cases, software is expected to operate seamlessly atop several different network infrastructures, which are switched between as available [21]. In all of these cases, the primary source of design-time uncertainty is the level and quality of network connectivity, with adaptation taking place to tolerate poor network performance and to best utilise faster, more reliable network connectivity when available.

There has also been much work into developing systems for autonomously managing, maintaining and best utilising so-called Quality-of-Service (QoS) in resource constrained networks (e.g. [22] [23]). In this field, software is expected to adapt to maintain

functionality in the face of the potentially volatile availability of network resources. When resources are scarce, applications deemed less worthy, either by their own labelling as in [22] or by dictatorship as in [23] of network resources have their access to the resources constrained or denied, but are expected to continue functioning to as great a degree as possible. The QoS field, from an autonomy perspective, is analogous to the network-centric systems field, in that adaptation is used as a means to mitigate network-originating uncertainty.

A degree of autonomy has emerged in the service-oriented systems field. These systems are particularly interesting because their use of adaptation isn't limited to tolerating network-originating uncertainty. Autonomous, service-oriented systems use so-called late binding [24] to allow units of work to be completed by different concrete services. Thus, a decision to offload a given unit of work to an alternative service, can result in a change in the overall system's observable behaviour. Using different underlying services may affect overall response time, accuracy or availability [25] [26], for example. The most promising approaches, from the perspective of autonomy, use brokers to establish and potentially negotiate requirements [27], select appropriate services and monitor requirement fulfilment [28] and switch underlying services to better achieve these requirements [29]. All of these tasks map well onto the traditional requirements engineering practices of requirements acquisition and negotiation, decision making, requirements monitoring [30], and our concern: adaptation.

Another area in which systems are increasingly expected to perform autonomously is that of fault-tolerant systems. These systems possess an ability to offer either complete [31] or partial [32] functionality in the event of some failure. Here, adaptation is used to replace, if continuing to offer complete functionality, or to operate without, if offering partial functionality, the failed component. Although this style of behaviour would typically fall within the scope of a moded system, it fulfils the definition of parametric adaptation, as discussed in Section 2.1. It should also be noted, that as both the software engineering practice supporting, and the technology underpinning compositional

adaptation improves, such functionality could be richer and more easily included using the more advanced adaptation style.

Although the systems developed for all of these domains differ slightly in the goals that each system's autonomy seeks to achieve; a commonality emerges: each of the systems adapts to enable it to better attain a certain software quality, be it speed, availability or reliability. These quality features would usually be represented at the requirements stage by one or more non-functional requirements (NFRs). A system's ability to attain the quality, or to fulfil the NFR, to any given degree is affected by changes in its operating environment, be this a change in available network resources, the poor performance of a service, or a software failure. In short, all of these systems adapt to best satisfy their NFRs in changing contexts, as posited in [33].

## 2.3 Existing Systems

There are several existing DASs that have been created, each using adaptation as a means to address specific problems. This section explores several of the systems highlighting how each effects adaptation, and how this adaptation helps the DAS to better fulfil its goals.

The first DAS examined is Lapouchnian et. al.'s adaptive image viewer [14]. This simple application was designed as a pedagogical DAS to highlight the complexity of the requirements engineering process in autonomous systems. The adaptive image viewer is a simple application for viewing images, whose only adaptive capability is to introduce or remove a caching component. Note that that such simplistic adaptive behaviour could easily be created using parametric adaptation, but the authors chose to use compositional adaptation so that the adaptive image viewer more closely reflected more complex DASs. The adaptive image viewer introduces the caching component in an effort to reduce latency when loading images for viewing, and removes it when the additional memory required by the cache can no longer be spared. The adaptive image viewer operates in a

relatively simple environment, with a simple Boolean variance: “Is there enough memory to use the cache?” The simplicity of the adaptive image viewer serves as a great asset when illustrating Software Engineering support or construction techniques for DASs. This thesis relies heavily on this example when introducing the ReAssuRE process in Chapter 4.

A good example of a service-oriented system that may be considered a DAS is Robinson & Kotonya's navigation system [34]. The navigation system was designed to showcase a quality framework for service-oriented systems, which makes an interesting if domain-specific adaptive infrastructure. The service broker, which the navigation system is designed to showcase, invites applications to request services including a specification of several quality features the application requires the underlying service to meet. These quality features are limited to a selection supported by the broker including latency, availability and cost. The specification is matched with the details held by the broker on a catalogue of available services, and if necessary the specification is negotiated to allow it to be fulfilled.

Limiting the quality features that an application may request a service to provide may seem restrictive, but doing so enables the broker itself to monitor service performance, and to renegotiate with or re-bind services to the application if performance falls below an acceptable level. It can be argued that all service-oriented systems operate in a relatively volatile context. When some unit of work is provided by an agent outside of the system's control it is natural to anticipate potential availability or performance problems, and adaptation, in the form of late binding in this case, can provide a useful means to mitigate this uncertainty.

One system illustrating this service and broker approach to adaptation is a navigation system. The navigation system relies on external services to provide location data, road maps, route calculations and points of interest to display on the map. Each of these services are selected at run time from a pool of available services, with quality features that fluctuate in line with network performance, service loading and the potential for random failure. The navigation system negotiates with the service broker, and thus with

the service providers, to obtain all of these services at as low a cost as possible whilst still maintaining acceptable performance and reliability.

Although the scope of Robinson and Kotonya's service broker is limited to service-oriented systems, it represents a novel approach to autonomy, and adaptation as a whole. The broker's ability to negotiate quality performance measures with both service providers and the system is analogous to design-time decisions made when engineering systems from scratch, and the performance monitoring aspect is near identical to requirements monitoring work proposed by Fickas and Feather [30].

There is some work that uses adaptation to maintain availability of service-oriented systems [25] [26] [35]. This work targets aggregate services, and in the event that one or more of the individual services upon which an aggregate service depends fails, the aggregate service adapts to replace the failed underlying service.

This section now examines two DASs performing the same function: a desktop e-mail client, and both sharing the same source of contextual volatility: different users. The first DAS, Fickas et. al.'s Think-and-Link e-mail system [36] was designed to be used by people with cognitive impairments due to brain injury. These users will differ greatly from each-other in ability to use the system, and the ability of each user may also change over time. The email system uses adaptation to adjust its interface to cater for different users as they use the system, according to profiles of each user's abilities provided by specialist doctors. The profiles themselves are created and updated off line, away from the email system. Some would argue that the off line creation of user profiles means the adaptation in this case is more analogous to maintenance and tuning. However, the email system adapts its behaviour when running to suit the differing needs of individual users automatically. Thus the creation and updating of a user's needs profile is more analogous to pre-design requirements elicitation.

The second DAS [37], performs a similar task for users without any specific medical needs. Lapouchnian et. al's DAS is built using the popular open-source e-mail client: Thunderbird [38]. The adaptive abilities of this email system are less radical, involving the

manipulation of options already present in the e-mail client to suit broadly specified user preferences. A user's preferences are expressed in terms of high-level goals such as “Prefers not to be interrupted” or the conflicting “Likes to answer e-mails quickly”. Models of these high-level goals are transformed into personalised configuration profiles, which Thunderbird uses to adapt, parametrically, its behaviour.

Despite the relatively low level of autonomy offered by the two e-mail systems, they share a particularly interesting trait. Each system's adaptive behaviour is derived directly from specifications of user requirements. The Think-and-Link system's user requirements are obtained clinically, from patients' doctors, whereas the second email system's user requirements are obtained from a high-level model. Each system's adaptive behaviour is derived from its requirements offline, with analysts manually creating appropriate configurations for individual users. By automating this process, and potentially allowing it to be carried out by the DAS itself, there may be an opportunity for greater autonomy. This is discussed further in Chapter 6.

Another good example of a DAS that has been built is Hughes et. al.'s GridStix system [2]. This DAS takes the form of an intelligent wireless sensor network that monitors the river Ribble in North-West England to predict flooding. The GridStix system operates in a hostile environment, with a significant probability of individual nodes failing due to submersion or physical damage from water-borne debris when the river floods. Furthermore, the monitoring location is remote, with battery-backed solar power being the only feasible option for powering the GridStix system. The tightly constrained power footprint combined with the expense in terms of power of providing sufficient fault tolerance to allow the GridStix system to operate in an adverse environment and to provide accurate flood predictions means that a delicate balance needs to be struck, with different fault tolerance mechanisms and flood prediction models used in different circumstances. The GridStix system adapts by enabling the more power-hungry mechanisms as the risk of flooding becomes more severe, and using more power-efficient ones at times of little risk. The GridStix system is discussed in more detail in Chapter 7.



The GridStix system uses adaptation as a means to mitigate contextual volatility, which originates from the river.

GridStix is interesting because the adaptation is both compositional, and performed autonomously. The technology underpinning the GridStix system [39] discussed further in the next section, is also fairly generally applicable, posing few restrictions on the type of environmental parameters that may be monitored, or the structural elements of the system that may be adjusted. As a result, the GridStix system is a short conceptual distance from the general case, with other domains easily imaginable in terms of different monitoring parameters and different software components.

One final DAS that has been proposed, but not yet implemented is Fraunhofer's Adaptive Assisted Living (AAL) system [40]. The AAL system controls a number of components throughout a house, aiming to ensure that the elderly resident is kept healthy by ensuring adequate, and appropriate, food and liquid intake, and summoning help in case of trouble. The AAL system uses sensors in the fridge to enable it to detect available food, sensors in cups to monitor fluid consumption and motion, orientation and pressure sensors to detect urgent medical problems. It is clear that in the AAL system, the major source of contextual volatility is the resident themselves, with it being almost impossible to fully understand and predict the ways in which the resident will live their life and interact with the AAL system. The proposed AAL system has been studied in detail [41], by focussing on identifying individual, and often independent, potential adaptations that the AAL system could use in different situations to ensure the resident's health.

The Adaptive Assisted Living system is interesting because of the large, open-ended nature of the potential contextual volatility. Any software engineering support or technology on which to base such a system needs to be able to handle a potentially large number of variations of environmental parameters and system configurations. Furthermore, with understanding of the AAL system's environment so incomplete, and with it being likely to remain so until well after deployment, the AAL system presents challenges in terms of ease-of-change, with flexibility and an ability to (re)tailor the system to the resident as its usage changes being of significant advantage.

## 2.4 Existing Adaptation Infrastructures

It is perfectly possible, and not uncommon, for a DAS to be created with no formal adaptation infrastructure, with the tasks of detecting contextual change, planning adaptation to suit and effecting the planned adaptation intertwined with the DAS's business logic. However, complex compositionally adaptive systems increasingly rely on some form of adaptation infrastructure dealing with adaptive behaviour, with varying degrees of adaptation and business logic segregation. The greatest segregation is typically found in the infrastructures in which the adaptive behaviour is explicitly handled by a separate software agent at run time. Approaches prescribing the design of a DAS conforming to some logical architecture remove the need for a run-time agent, but do so at a cost of removing the clear distinction between adaptive behaviour and business logic afforded by the multi-agent approaches. Almost no separation is afforded by approaches seeking to add explicit adaptation support to programming languages, either by retrofitting existing, or developing new, specialised, languages. Each of these approaches is discussed more fully in the following subsections.

### 2.4.1 Multi-Agent Adaptive Infrastructures

The term “multi-agent adaptive infrastructure” encompasses any middleware-based approach, in which the middleware itself can be considered an independent software agent performing adaptation-related tasks. The term also covers any broker-based infrastructure, such as one underpinning many a service-oriented DAS. A multi-agent adaptive infrastructure typically offers the greatest degree of separation between adaptive behaviour and business logic, and also offers a means to reduce the complexity of the adaptation logic by offering an existing agent capable of performing contextual monitoring, adaptation planning and effecting off the shelf (for example, [42]).

There has already been some research effort on creating service-oriented architectures with support for dynamic adaptation (e.g. [42], [43] & [44]), most typically aiming to support QoS controls on multimedia streaming over a wireless or ad-hoc network. In addition, there has also been work to develop a self-managing, adaptive content distribution network that transcodes video & audio streams and reorganises itself dynamically at runtime in response to the environment, and network conditions [45]. Each of these infrastructures offers an agent to effect adaptation, and some provide means to monitor contextual change. None of these service-oriented architectures, however, offers support for the planning of adaptation. DAS developers using one of these architectures would still need to develop code explicitly requesting that the broker performs a specific adaptation in specific circumstances. This weakness is not inherent to service-oriented architectures, with Robinson and Kotonya's service broker performing the task autonomously.

There are also a small number of adaptive service-oriented architectures that plan and effect adaptation to maintain the availability of an aggregate service amidst the potentially shifting landscape of available underlying services [25] [46] [35]. Each of these infrastructures refers to itself as self-healing, and is limited both in domain; service-oriented systems, and in purpose; adapting to improve availability. Additionally, there has been work [47] offering an infrastructure to allow a service-oriented system to adapt to minor changes in requirements.

There has been significant research into using adaptive middleware for supporting QoS in multimedia distributed systems (e.g. [48] [49] [50]). There has also been some research effort [51] on developing adaptive middleware to support combat systems such as self guiding “smart” bombs and automated intelligence gathering aircraft. Although the architectures differ in the level of support offered for all the identified adaptive behaviour, each is domain-specific.

One of the first examples of a generally applicable dynamic middleware was DynamicTAO [52], which is a CORBA Object Request Broker (ORB) retrofitted with the ability to utilise different strategies to support “concurrency, request demultiplexing,

scheduling, and connection management” [52] specified at load-time. Although this behaviour is essentially statically adaptive rather than dynamically adaptive, the work led on to UIC-CORBA [53], which allows abstract components to be created at design time, specialisations of which are dynamically loaded at runtime. Each infrastructure offers a means to effect adaptation, but offers no support for contextual monitoring or adaptation planning.

The OpenCORBA ORB [54] is another CORBA ORB that allows proxies to select remote objects to invoke dynamically according their own logic. This dynamic invocation facilitates the swapping of compatible objects at run time. However, as with UIC-CORBA, there is no support for contextual monitoring or adaptation planning,

Lancaster's OpenCOM reflective framework [55] bears similarity in terms of adaptive functionality to UIC-CORBA, although it is not a full middleware in itself. OpenCOM is more of a component-based programming model, which allows components to be loaded and unloaded, bound and rebound at runtime. Essentially, OpenCOM, OpenCORBA and UIC-CORBA offer similar means to effect adaptation, with the same lack of support for the other aspects of adaptive behaviour.

Built atop OpenCOM is Gridkit [39], a middleware designed for grid computing. Gridkit is designed to run on a wide variety of devices, from wireless sensor networks through to PDAs and desktop computers. Adaptation in Gridkit is achieved by swapping out “Component Frameworks” and thus individual components, in configurations specified in *policies* written in XML. Crucially, specified in the policies, are also the events that are to trigger adaptation. Thus, Gridkit's adaptation policies codify the DASs adaptation planning, reduce the complexity of contextual monitoring to the firing of events, and OpenCOM's ability to effect adaptation is extended.

Typically, a DAS constructed using an infrastructure providing a distinct agent to handle adaptation specify the behaviour of the adaptive agent using adaptation *policies* (e.g. [39] [56] [57] [58]). These policies are written as Event-Condition-Action rules [59],

stating that a given adaptation is to be performed when a specific event is raised under certain conditions, although a more expressive policy language may emerge in the future.

## 2.4.2 Architectural Adaptive Infrastructures

Some, notably Kramer and Magee [3] argue that although adaptive middleware is helpful in composing a DAS, it lacks generality outside of the distributed systems domain, and that adding language support for adaptation offers no abstraction and thus does nothing to aid DAS design. More general approaches make use of a pre-set, generic, transferable architecture separating different aspects of adaptation logic from the application's business logic.

Kramer and Magee's approach borrows heavily from robotic system architectures, following a three tier approach, as depicted in Figure 1. Each component is expected to fine-tune its behaviour at the lowest level, the middle level implementing adaptation based on strategies devised by the upper layer, which uses goal based reasoning.

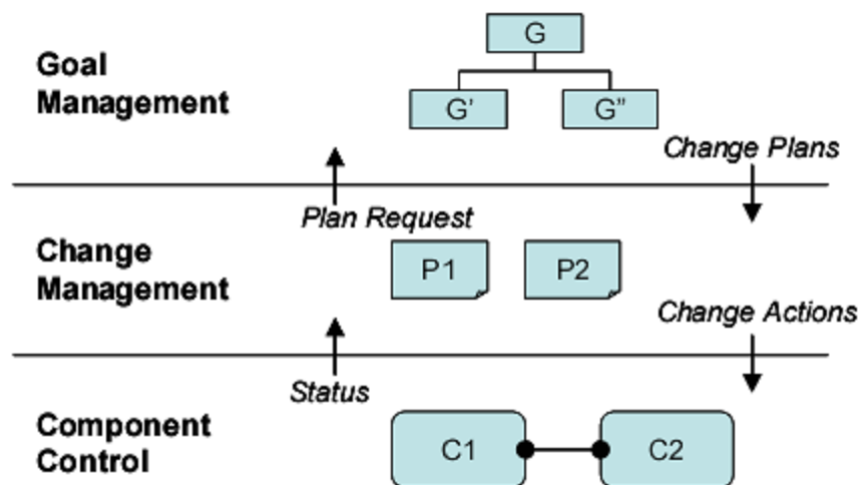


Figure 1: Kramer and Magee's Adaptive Architecture [3]

It is clear that the architecture in Figure 1 is well designed for the tasks of planning and effecting adaptation. The task of detecting contextual change, however, is seemingly left to individual components. Although individual components may be well placed to monitor their own performance, monitoring of the environment as a whole, or some specific aspect thereof, may be better performed at the system level. This approach offers a useful template for DAS design, particularly if the system's adaptive behaviour is aimed at tuning.

Oreizy et. al. [6] argue that a software architecture model of the components, connectors, inputs and outputs of a DAS could be used by some sort of adaptive manager to plan and effect adaptation. The models are produced specifically for this purpose, and are updated by the DAS's adaptive infrastructure at run time. Although neither the adaptive manager nor the infrastructure connecting components and updating the model have been created, the use of a model to guide adaptation offers the possibility of separating the tasks of planning and effecting adaptation from the business logic in the individual components. It may also be possible to re-use an adaptation manager using this approach across domains, if the architectural models used are not too domain specific.

White et. al. [60] focus on how individual components of a DAS should adapt their behaviour in response to established policies and should interact with one-another. They also look at how an adaptive infrastructure could allow the individual components, termed “elements”, to find, monitor, aggregate, broker and negotiate between themselves. This work promotes a more weakly-coupled, less systematic approach than the previous two, with even greater control placed in individual components. It is easy to imagine a system designed in such a manner offering a relatively high degree of autonomy but, unfortunately, the level of complexity of both the adaptive infrastructure and the components places systems using the architecture out of reach at present.

An architectural approach to DAS design essentially replaces a concrete adaptive infrastructure with a template by which a bespoke one could be designed to suit a specific DAS. Although an architectural approach typically offers a good degree of separation between the DAS's adaptive behaviour and its business logic, the motivating concern

behind the need for separation: explosive complexity, is ignored. A multi-agent adaptive infrastructure offers the possibility of agent reuse, allowing the complexity associated with adaptive behaviour to be dramatically reduced. It is for this reason that we are seeing actual DASs created using multi-agent adaptive infrastructures rather than designed to abstract architectures.

### 2.4.3 AI Planning for Adaptation

Several of the adaptive architectures discussed in the previous subsection rely on some planning component to plan and devise adaptations to be effected. For example, the “Goal Management” component in Figure 1 is responsible for devising change plans for the “Change Management” component to effect. AI planners [61] offer the most promising possibility for implementing such a planner, and could be use outside of a formal adaptation infrastructure to plan adaptations in autonomous systems in general.

AI planners typically operate on sets of propositional formulae describing the initial, or current, state of the world, some description of a desired state the planner should seek to attain, and some description of available actions. The planner's output is a sequence of actions, which when executed from the world's initial state will yield the desired state

One notable AI planner is NASA's Livingstone [62], which forms part of NASA's work on so-called remote agents [63], which offer autonomous control of a spacecraft for extended periods of time in a volatile, hostile, space environment. Livingstone uses component-based declarative models to provide adaptive capability to allow fault diagnosis, recovery and tolerance. Hardware failures aboard spacecraft are common, and a successful mission depends on the spacecraft being able to withstand multiple component failures, potentially interrelated.

The Livingstone planner depends upon formal models of a spacecraft's systems, and of expected behaviour in all valid configurations. Plans are selected from those available on a least-cost basis, where cost is expressed in terms of the expenditure of finite resources, for

example propulsion fuel, or the use of irreversible actions, for example one-time explode-shut valves. Thus, the Livingstone planner requires detailed knowledge of a DAS's expected performance in a given configuration, and of the costs incurred in switching, or maintaining a given configuration. This thesis' work concerns the use of requirements models for DASs, which may be considerably less complete than the models required by Livingstone. However, the ability to select configurations, and suitable adaptation plans to transition to a new configuration from a set of those available is conceptually similar to the model-driven adaptation discussed in Chapter 6.

#### 2.4.4 Explicit Programming Language Adaptation Support

By adding explicit support for adaptation, a programming language can go some way to reducing the additional complexity involved with a DAS. Although the use of an adaptation-supporting programming language does not remove the need to explicitly code the DAS's adaptive behaviour: detecting contextual change, planning adaptation to suit and effecting the planned adaptation, it can reduce the workload associated with one or more of these tasks.

Program Control Language, or PCL [64], is a set of language extensions for both C++ and Java targeted at allowing application programmers for distributed systems to specify both when and how applications should adapt at runtime. The extension provides extra language primitives, termed “adaptors” to allow classes to be swapped at runtime, values, termed “metrics” to be monitored during execution” and events that may trigger adaptation. These primitives can be used to reduce the complexity of detecting contextual change and of effecting adaptation, but offer little assistance in planning adaptations in response to detected contextual change.

Adaptive Java [65] augments the standard Java language primitives with additional keywords to support the observing of runtime behaviour, i.e. introspection, and changing this behaviour, i.e. intercession. The language is targeted at adaptive middleware



developers, much like OpenCOM [55], differing with the adaptive functionality encapsulated within the language rather than an additional library. The language offers no additional support for detecting contextual change or for planning appropriate adaptation. It does, however, offer considerable support for effecting said adaptation.

Although Trap/J [66] and Kava [67] can be classed as language support, they are targeted at augmenting existing Java applications with adaptive properties after deployment, without modifying their source code. Although one would expect this “clean” modification to a Java class' source code to require a modified Java Virtual Machine (JVM), each approach adds adaptive capability by selecting classes that are to be adaptable at compile time, and uses a special compiler to generate reflective byte code associated with the selected classes. at run time, interfaces implemented by the newly modified, adaptable classes may be used to modify system behaviour. Of all the programming language based approaches discussed in this section, this shared approach offers the greatest degree of separation of a DAS's adaptive behaviour and business logic. The original system remains untouched, with an additional potential adaptive capability added at compile time. However, code to detect contextual change, and to plan the newly-enabled adaptation still needs to be created, external to both processes. As such, despite the relatively high degree of separation, Each of Trap/J and Kava only mitigates the additional complexity of effecting adaptation.

## 2.5 Chapter Conclusion

Despite the relative complexity of DASs serving to limit their adoption to scenarios in which they will yield most benefit, DASs are being created to serve a variety of purposes. The domains for which DASs are being built are characterised by volatile changeable conditions. Different modes of operation are sometimes possible only in some conditions, and the relative priority, and expected degree of satisfaction, of requirements varies in tandem with the environment.

Although there have been several (e.g. [65], [66] & [67]) attempts to create programming languages with built in support for adaptation, there has been little adoption of these languages. In a similar vein, there has been little adoption of the several adaptive architecture designs proposed (e.g. [3] & [60] by which DASs could be constructed, and adhering to one of these architectures could be eased with the use of an adaptive programming language. Instead, the DASs actually being developed (e.g. [34] & [2]) are using an independent agent to handle adaptation, in the hope that the agent can be re-used, and to simplify development by separating the concern of adaptive behaviour from the DAS's business logic.

Returning to the research aims discussed in Section 1.3, we can see that one of the questions this thesis aims to answer is as follows:

To what extent is a DAS's adaptive behaviour merely a derivation of environmental analysis and configuration decisions?

Section 2.4.1 discusses the fact that many a multi-agent adaptive infrastructures, or more specifically adaptive middleware, typically utilises adaptation policies to codify a DAS's adaptive behaviour. It is therefore possible to demonstrate a link between the environmental analysis done, and decisions taken for a DAS by showing it possible to derive an appropriate adaptation policy from the analysis.

Despite the emergence of DASs using some form of adaptive infrastructure to handle their adaptive behaviour, and the benefit of separating this adaptive complexity from the DAS's business logic, the specification, design and construction of DASs remains extremely difficult. The next chapter reviews the state of the art of requirements engineering for DASs, which faces a particular challenge in specifying variable or multiple behaviours, when dealing with a complex and potentially incorrectly or only partially understood operating environment.

## 3 DAS Requirements Modelling & Monitoring

---

This chapter presents a somewhat narrower literature review than Chapter 2, focussing on requirements modelling and requirements monitoring, two key and intertwined aspects of DAS research.

One of a DAS's defining characteristics is its ability to monitor the environment, enabling it to adapt its behaviour to better suit the prevailing operating conditions. It is not, however, always possible to monitor the environment directly, and one source of monitoring data may not be sufficient to reliably infer the current status of the operating environment. In some cases, it is possible to monitor the degree to which a DAS's key requirements are being satisfied, and use this data as a surrogate for more direct environmental knowledge. In addition to this “surrogacy” use of requirements monitoring, it can be argued that [33] fundamentally, all DAS adaptation is motivated by a desire to strike a balance between conflicting requirements in changeable and uncertain environmental conditions. From this viewpoint, requirements monitoring becomes a cornerstone of the enabling technology for DASs.

If so much of a DAS's behaviour is motivated by the need to balance conflicting requirements in changeable conditions, then it follows that in order to understand, design and build a DAS it is necessary to have a complete understanding of the requirements, and what constitutes an acceptable balance in different conditions. It is this problem that requirements modelling for a DAS aims to address, with a strong consensus emerging that goal-based requirements modelling methods are best suited to modelling the complex, variable systems that attract the DAS moniker.

The following subsections discuss requirements modelling and requirements monitoring in more detail, examining existing work applied to DASs.

### 3.1 Requirements Modelling

At the most general level, Berry *et. al.*'s work studying the type of RE done for (and arguably by) DASs separates the types of RE into *levels*, with the more “meta” RE activity done at the higher levels. The work doesn't discuss how each level could be modelled. However, the level separation forms the basis of the modelling method discussed in Section 3.1.2, and as such is discussed here in preparation.

Berry *et. al.*'s first level RE resembles the standard practice for non-adaptive systems, and involves analysing the proposed DAS's operating environment, specifying the functionality of the DAS as a whole, and crucially identifying all the potential configurations the DAS may adopt, and analysing the circumstances under which each should be adopted. The work terms the individual configurations *target systems*, and the circumstances under which they are to be adopted *domains*. For ease of understanding, this can be conceptualised as the operating environment being *partitioned* into several domains, with a target system devised and specified for each.

Second level RE looks at how the DAS should distinguish between domains, and how to identify the appropriate target system for each. If Level-One RE is concerned with individual target systems, and how they are to behave in their given domain, then Level Two RE is concerned with the DAS as a whole, and how the target systems are amalgamated into the complete system. This is the level at which it can be argued that the DAS itself is performing RE, but this thesis is concerned with only the activity undertaken.

Third level RE looks at how the Level Two RE may actually be accomplished. It involves specifying the mechanisms by which the data needed to distinguish between domains may be obtained, and the mechanisms by which a target system deemed appropriate may actually be adopted. Given that a DAS often relies on some form of adaptive infrastructure to perform environmental monitoring and to effect adaptation,

Level Three RE often involves analysis of the capabilities of available adaptive infrastructures.

Fourth level RE is the furthest removed from DAS behaviour, covering the specification, development of and research into adaptation mechanisms in general. Typically, the adaptation mechanism developers and the DAS developers will not be the same people, and thus the Level Four RE occurs separately from the other three levels.

The majority of the research into DAS modelling focusses on the Level-One modelling described above, and sometimes only offers partial coverage for the Level-One RE. For example, Morandini *et. al.*'s work [68] on extending the Tropos [69] methodology involves the creation of goal models to analyse the potential impact of the DAS adopting different solution strategies, but doesn't offer any explicit means of modelling the circumstances under which each may be preferable. Furthermore, environmental monitoring and the mechanism by which different solution strategies are adopted are left completely external to the modelling process. Other DAS modelling approaches can be clustered by the underlying modelling techniques utilised, with work having been undertaken using both the KAOS [70] and i\* [71] modelling techniques. The work in each cluster is summarised and discussed in Sections 3.1.1 and 3.1.2, respectively.

The research goal identified in Section 1.3 of attempting to derive a DAS's adaptive behaviour directly from requirements models may seem distant, but attempts have been made to derive other aspects of system behaviour and configuration from goal models previously, with some success. Yu *et. al.* Have shown it possible [72] to derive feature models and component interfaces from generic goal models created using the OpenOME modelling tool [73], whilst Letier and van Lamsweerde have developed a method [74] to derive operational specifications directly from KAOS goal models. Neither of these two pieces of work focus on the derivation of adaptive behaviour from goal models, but do show research interest in deriving lower-level artefacts using information contained in relatively early-phase, high-level goal models.

There has also been some work looking into how best to write the requirements specifications for a DAS. In RELAX [41] prescriptive and absolute modal verbs are replaced with fuzzier, more malleable statements about the goals the system endeavours to fulfil to as great a degree as possible. Requirements in the form of: “The system shall do  $x$  as early as possible after  $y$ ” offer a degree of flexibility in how the system chooses to fulfil them, and offer the possibility of trading-off less complete fulfilment of one requirement for more complete fulfilment of one or more others. The approach is underpinned by formal semantics, essentially bridging between precise formalisms and high-level goal modelling. This approach is useful for recording requirements that the system as a whole attempts to best fulfil, and may also be useful for describing adaptive behaviour. However, if it is possible to derive a DAS's adaptive behaviour directly from models created earlier during the RE process, this description may be unnecessary.

There has been some work that seeks to allow requirements models that encompass estimated values for non-functional properties to be updated at run time with observed values from a DAS [75]. Unusually, [75] uses Discrete Time Markov Chains (DTMCs) [76] as the requirements model, and the functional properties being monitored apply to service-oriented systems. The work mentions the possibility of a DAS reconfiguring itself in response to changes in the model which, if exploited, would yield a model-Driven Adaptive System (m-DAS), as discussed in Chapter 6.

Goal-based requirements modelling methods are attractive for DAS modelling, because of their inherent ability to reason about alternate means of satisfaction for individual goals, and the ability to reason about different ways of breaking down, or *operationalising*, high-level goals into more precise, lower-level goals, which would typically illustrate different solution strategies. This natural ability to analyse several different possible solutions lends itself well to a DAS, which may well adopt several of the possible solutions in different circumstances.

### 3.1.1 KAOS-based DAS Modelling

KAOS [70] is a structured, goal-based requirements modelling method, that pays particular attention to softgoals. Softgoals are often early manifestations of Non-Functional Requirements (NFRs), which are often overlooked in other formalisms. NFRs are particularly important for a DAS, given (as argued in [33]) that striking a balance between conflicting NFRs in a variable environmental conditions is a key driver in the decision to create a DAS over a non-adaptive system.

KAOS has a relatively well defined formalism, covering goals, softgoals, and domain assumptions which will prove an advantage in performing model analysis and derivation. The notion of *obstacle* modelling, which promotes reasoning about potential pitfalls and impediments to goal satisfaction could also prove useful in modelling foreseen environmental uncertainty, and in identifying areas in which uncertainty exists. However, the role of formalism in KAOS also hampers the process of changing models as understanding improves, by requiring change at both the model and formal level. Given the complexity of the environments for which DASs prove most useful, and the likelihood of inaccuracies and incompleteness in and of the understanding of the environment, a significant workload in making model changes amounts to a substantial limitation. Incomplete knowledge of the environment also serves to limit the benefit of obstacle modelling, given that the impact only of (at least partially) foreseen obstacles may ever be analysed.

There has been some work [77] [78] looking at ways in which individual adaptations could be represented using KAOS, with Brown *et. al.* adding an additional linear temporal logic (LTL) operator to the KAOS formalism to explicitly capture a change in the requirements a DAS is to satisfy. The work builds upon earlier work [79] in establishing LTL patterns for common adaptations (such as one-point adaptation, where one target system stops execution, transfers state and a second target system commences execution), with [77] essentially incorporating the patterns into KAOS models. Although the possibility of deriving a DAS's adaptive behaviour (the concern of switching between

behaviours in appropriate contexts) from the created goal models is identified, no method to do so has been developed, with the work's focus being the specification of adaptive behaviour. This work is unusual in focussing solely on the second level RE (as mentioned in Section 3.1), with Level-One RE performed on a per-target system basis external to the process.

There has been some work [80] also into the derivation of architecture models for a DAS directly from KAOS goal models. The work explicitly highlights goal conflicts as the motivator for adaptation (as in [33]), but unlike Brown *et. al.* [77] omits support for representing a DAS's adaptive behaviour, focussing instead on deriving an architecture that would support adaptive behaviour specified and created elsewhere. Furthermore, the generated architecture model omits one of the crucial tasks in Level-One RE: analysis of what behaviour should be adopted under which conditions. This prevents the work from being combined with the previously discussed adaptation modelling to allow a DAS's adaptive behaviour to be derived from a set of KAOS goal models.

Finally, there has been some work seeking to add support for reasoning about partially satisfied softgoals to KAOS [81]. In practice, the softgoals that may only be partially satisfied are goal-modelling representations of NFRs, for which it is difficult to establish solid satisfaction criteria. This work is important, as DASs typically operate in environments that prohibit the complete satisfaction of all the DAS's goals, - and volatility in the environment prevents a single acceptable balance between them from being struck. The work proposes that domain-specific objective measures of goal satisfaction are proposed, and when deciding between different candidate goal satisfaction strategies, a quantitative analysis of the options is performed using stochastically modelled, typical or estimated data to reach a decision. Of course, the accuracy of the method is limited by the accuracy of the underlying data, but the work does note that it may be possible to monitor the data, and in a DAS it may be possible to adopt an alternate candidate if analysis of actual measured data no longer supports the adopted goal satisfaction strategy.

To conclude, although there hasn't to date been any all-encompassing work in modelling a DAS completely in KAOS, there has been a significant amount of work



looking to model specific aspects of a DAS's behaviour. The behaviour of individual target systems has been modelled in [80] with adaptations between target systems modelled in [77]. DAS modelling support could be enhanced by including the partial satisfaction reasoning method introduced in [81], allowing balanced “best fit” configurations to be designed for different target systems in each domains. There is also a possibility of designing a DAS with greater autonomy by granting it the ability to monitor the data supporting the selection of “best fit” configurations, and re-making the selections in the event of the data being demonstrably incorrect at run time.

### 3.1.2 i\*-based DAS Modelling

i\* [71] (pronounced I-star, and meaning “distributed intentionality”) is a goal-based and agent-oriented graphical modelling methodology aimed at supporting high-level reasoning during early-phase requirements engineering. There are several related methodologies that extend or specialise i\* for specific purposes, all (like i\*) created at the University of Toronto, for example: GRL [82] and Tropos [83]. By allowing a more complete means of modelling, understanding and reasoning with the DAS, its constituent parts and the environment *in situ*, i\* seeks to enable thorough exploration of potential solutions, and their potential consequences of their selection. There are two types of i\* model: Strategic Dependency (SD) and Strategic Rationale (SR). The two models are used for slightly different tasks

In SD models, systems are modelled in terms of *agents*, *goals* and *dependencies*, with models offering a high-level view of which parts of a (socio-technical) system depend on which others to function. In i\* terminology, an agent is either a human or automated system component, a goal is something an agent seeks to achieve, and a dependency runs from one agent (the *dependor*) to another (the *dependee*) via the subject of the dependency (the *dependum*). There are other elements that may appear on SD models, for a full description see [71].

In SR models, the rationale behind a goal satisfaction strategy may be analysed in terms of *tasks*, *softgoals* and *contributions*. A task may be *decomposed* into one or more sub-tasks, and tasks also serve a means to satisfy goals, when attached via a *means-end* link. Softgoals are an important concept, representing a goal with no clearly defined satisfaction criteria: essentially an early representation of an NFR. The completion of a specific task may have one or more impacts on softgoals, captured with *contribution links*. There are several different types of contribution link corresponding with the magnitude and polarity of the captured impact. Selection of one task over another (that satisfies the same goal) implies an acceptance of, or at least a preference for, its softgoal impact(s).

There have been several general-purpose i\* modelling tools developed [84] [85], which ease in the construction and validation of a SD or SR model. Some tools offer some limited support for reasoning with a finished model, and this tool support can be leveraged to ease requirements modelling for a DAS, as much as any other class of system.

There has been some work [86] [16] looking to model DASs using i\*. The LoREM process advocates creating i\* models embodying the level 1-3 RE (according to the levels defined in [10]) described in Section 3.1, allowing analysts to study a DAS as a whole, reason about the configurations of each target system, understand the adaptations involved in switching between target systems, and to understand what is required of the adaptive infrastructure underpinning the DAS. The models created at each level are discussed below:

At Level One, the method involves creating an SD model of the DAS as a whole, illustrating the DAS in context, showing how it fits in with the environment, and which stakeholders depend on the DAS to do what. Also at Level One, an SR model is created of each individual target system, showing the configuration the DAS adopts and how it balances conflicting softgoals in each domain. There has been additional work [87] [88] augmenting these Level-One SR models with *claims*, which were first seen in the NFR framework [89]. Claims make explicit the rationale behind a decision, with finer granularity than simply recording a series of softgoal priorities in each domain. The use of claims is discussed further in Sections 4.2.1 and 4.2.2.

At Level Two, the method involves creating an SR model of each valid transition from one target system to another. The model shows how the DAS' environmental monitoring mechanism, the decision making mechanism and the adaptation mechanism work together to achieve a specific adaptation, and shows the circumstances under which the adaptation will be made.

At Level Three, the method involves creating an SR model of each of one or more prospective adaptive infrastructures for the DAS. The DAS depends on the adaptive infrastructure to provide the monitoring mechanism, decision making mechanism and adaptation mechanism referenced in the Level-Two models, and it is here that these are analysed. Each adaptive infrastructure will have different abilities to fulfil these roles, and the choice of adaptive infrastructure could have far-reaching implications. At present, the relatively small number of available adaptive infrastructures tends to mean that a DAS is envisaged with a particular adaptive infrastructure in mind: in this case, the model serves as a means to reason about the limitations imposed by this choice. With a more developed ecosystem, the utility of the Level Three modelling will be in comparing different prospective adaptive infrastructures.

There has been some work [90] also attempting to derive requirements from i\* SD models. By observing patterns in the dependencies between actors, it is possible to derive (candidate) textual requirements for the dependee to fulfil the dependency, and for the depender to expect the dependee to do so. In addition, some NFRs can be generated in some circumstances, typically accuracy and timeliness NFRs in [90] – with the type of requirements generated reflecting on the air traffic control domain used as case study. It may be possible to generate other types of NFR in domains in which they feature more prominently. The derivation in this work [90] is performed manually, although automation is suggested.

Finally, there has been some work [91] looking to derive use cases directly from i\* models. Although this thesis is concerned with deriving adaptive behaviour from models rather than use cases, Maiden *et. al.* [90] and Santander *et. al.* [91] show that there is research interest in using i\* models as a source to derive lower level artefacts from.

To conclude, there is already some notable work [86] that uses i\* as a basis for DAS modelling, with the combination of goal and agent orientation considered useful for high level reasoning during early-phase RE. There is also some research interest [91] [90] in deriving lower level artefacts from i\* models, as suggested with respect to deriving a DAS's adaptive behaviour in Section 1.5.

## 3.2 Requirements Monitoring

Requirements monitoring is the recording and gathering of information from which it can be determined the degree to which a system is meeting its requirements [30]. This is a key ability, underpinning the capability of a DAS to tailor its behaviour to a volatile environment. It is the ability to identify that the current system configuration isn't fully meeting the requirements, along with the ability to identify a closer to optimal configuration that allows adaptation to occur. Even when the possibility of performing requirements monitoring was first raised, the idea of “realigning” the system in response to the monitoring data was floated and it is arguable that this idea represents the origin of the DAS as a class of system.

Feather *et. al.* [92] envisaged systems that monitor their requirements satisfaction, and adopt an alternate goal satisfaction strategy in the event of the system failing to satisfy one or more requirements. In this work, the goals and satisfaction strategies were modelled using KAOS, with code to monitor system performance being created, and gathered data analysed to establish whether reconfiguration is necessary. Although the behaviour described is clearly similar to (and a subset of) DAS behaviour, there are some key differences. These are discussed in the following paragraphs.

Firstly, the systems described by Feather *et. al.* don't possess any knowledge of their operating environment, and don't seek to detect changes in the environment. A failure to satisfy a requirement could be caused by an unexpected change in the environment, a deficiency in the system design, or a system fault; all are treated identically. The authors

do, however, note that the system's inability to meet its requirements as expected could indicate that the system is being used outside of its designed operational envelope.

Secondly, the systems described by Feather *et. al.* have no knowledge of which potential alternate configuration is likely to perform best in any given scenario: the work describes systems that have just one alternate configuration, and it is assumed that this configuration will better suit *all* circumstances in which the first proves inadequate.

Combined, these leave the systems described by Feather *et. al.* possessing only some of the adaptive capabilities assumed of a DAS. Essentially, the described systems are a DAS operating in only two domains: *expected* and *unexpected*. The DAS adopts a fixed configuration in the expected domain, and any inability to meet its requirements is taken to mean a switch to the unexpected domain, in which a second configuration is used. A useful analogy to this style of DAS is the “Limp Home” mode seen in automotive systems, which allows automobiles to function sufficiently to get to a garage (or to get home) even in the event of a fault.

The tricky decision of identifying requirements that the system could benefit from monitoring is addressed by Robinson [93], who focusses on monitoring *obstacles* at runtime. Obstacles are part of the KAOS ontology, and represent a condition that prevents or hinders the satisfaction of a goal. For example, someone's efforts to clean their windows may be hindered by the unavailability of hot water, thus their “Clean Window” goal faces a potential obstacle of “No Hot Water”. The KAOS research community have worked on the issues of the identification of, and reasoning with, obstacles [94], and Robinson argues that it is the presence of these obstacles that should be monitored either directly or via some surrogate property.

Robinson's work focusses on monitoring aggregate web services, which the authors correctly argue offer the possibility of reconfiguring to use a different combination of underlying services in response to the presence of an obstacle. Such a system would certainly be classified as a DAS, with the configuration adopted in the presence of each combination of obstacles (including the scenario without obstacles) being equivalent to a

target system, and snapshots of the operating environment with each combination of obstacles being equivalent to domains. A related and more general approach is presented by Robinson [95], which applies the same concepts to traditional, non-adaptive systems.

The main weakness of the obstacle-driven process is that only those obstacles foreseen and monitored for can be analysed and have a configuration devised for. Although the problem can be mitigated using very broad obstacles that cover several different problem scenarios. The use of broad obstacles may be beneficial, given that careful grouping of specific obstacles under broader obstacles on the goal model could yield a reduction in the number of target systems that need to be created. However, there will always be a risk of an unforeseen obstacle preventing a goal being satisfied, and that the system will have no means of recovery.

Building upon the requirements monitoring literature are the related areas of requirements reflection [96] and requirements-aware systems [97]. Requirements reflection proposes requirements that can be represented at run time, and the satisfaction of which can be monitored directly rather than via some surrogate property. A requirements-aware system is a system that maintains some run-time representation of its own requirements, in order to reason with them (and trade-off conflicting requirements) in changeable contexts at run time, using self-adaptation as the means to effect changes identified as appropriate by the reasoning. The use of a requirements model at run time as suggested in Chapter 6 is one method by which a system could achieve requirements awareness, or be considered to be reflecting upon its requirements.

To conclude, requirements monitoring is a key enabling technology, underpinning the ability to create DASs, with the requirements monitoring literature [30] [93] identifying adaptation as a key potential benefit of runtime monitoring. There is a strong consensus that the use of goal modelling allows the need for, and consequences of requirements monitoring to be analysed, with several KAOS-based modelling approaches being devised [77] [80]. The goal-based approaches proposed each use obstacle analysis to identify properties to monitor, with the presence of obstacles being monitored in lieu of some direct satisfaction metric for a specific requirement. Obstacle monitoring trades a fine-

grained ability to analyse specific “what if?” scenarios with the ability to react to unforeseen problems in satisfying requirements. By monitoring other environmental properties, it may be possible to better adapt to less foreseeable conditions.

### 3.3 Chapter Conclusion

There is a strong research consensus that identifying and striking balances between conflicting requirements is a key feature for any DASs, with Berry *et. al.* [10] going so far as to argue that a DAS itself is performing some form of RE at runtime. At the very minimum, a DAS is reasoning with, and adjusting its ability to satisfy different requirements, if not monitoring them either directly or indirectly [92] [93]. Despite RE's crucial role in supporting DASs, RE support for creating DASs is limited, and a need to improve RE support for DASs has been identified as a notable research challenge by Cheng & Atlee [9].

There is also a strong consensus that goal-based requirements modelling is well suited to DAS RE activity, given the inherent ability to reason about alternative decision choices. This ability is particularly useful for DASs, which may well adopt each of the alternative choices in certain circumstances, as opposed to non-adaptive systems for which a decision would be made during the specification process, and only ever revisited in performing maintenance. As such, there has been a significant amount of research effort looking at the use of goal-based requirements modelling for DASs [77] [80] [86]. Some techniques focus on modelling potential adaptations, and scenarios for their use individually (e.g. [77]), whilst some package several adaptations into a balanced strategy for a foreseen set of environmental conditions (e.g. [86]).

One of the research questions raised in Section 1.3 is:

How can a system be designed with a greater degree of autonomy than current state-of-the-art DASs, and is the extra autonomy useful?

There appears to be a limitation in some of the goal modelling techniques used to model DASs, perhaps dating back to the modelling performed in requirements monitoring literature [92] [93]. Techniques based around an obstacle-modelling paradigm are limited to analysis only of known and foreseen obstacles. Techniques based around domain partitioning face a related problem, in that they are limited to analysis only of known sets of environmental conditions (or combinations of monitored environmental properties). To offer a DAS with a greater degree of autonomy, it will be necessary to at least partially overcome one of these limitations.

Several pieces of the surveyed literature [92] [30] raise the possibility of a system notifying stakeholders that it is operating outside of its designed operational envelope. However, save for the recent requirements-aware systems literature [97], there is little discussion of actually taking action in such circumstances. DASs offer the possibility of taking corrective action when encountering an unforeseen environmental condition, be it characterised as an unidentified obstacle, as a faulty assumption, or as an unexpected combination of individually foreseen conditions. Furthermore, the chance of such a situation arising is significant, given the complexity and relatively uncertain nature of the environments for which DASs prove most beneficial. As such, this thesis seeks to go some way towards demonstrating how a DAS can adapt when faced with unforeseen environmental conditions.

At this stage, it is clear that some form of goal modelling would be useful for the research, in that, with suitable extension, it would be possible to use goal models to record assumptions made and decisions taken, and offer sufficient traceability to later identify those that combine to specify the DAS's adaptive behaviour. Furthermore, it may be possible for a DAS to reason with a goal model at run time to guide its adaptation. It is also clear that extending some of the existing work modelling DAS requirements with either KAOS [70] or  $i^*$  [71] would be appropriate. The potential benefits of each technique are discussed in the following paragraphs.

Using the KAOS-based technique [80] would bring with it KAOS's well defined formalism, which may offer a more direct approach to offer a run-time representation of a



requirements model. [80] also offers the useful obstacle modelling activity, for which there is no equivalent in [86]. However, the level of integration of formalism in KAOS is also something of a concern, given the likelihood of models undergoing potentially rapid and iterative change during the RE process due to the relatively poor understanding of typical DAS operating environments. Maintaining accurate, verified, and up-to-date formal models of a DAS's expected behaviour during early-phase RE could prove burdensome, and the models constructed are relatively complex, given KAOS' lack of built-in agent orientation, which provides a useful means of abstracting over individual DAS configurations by declaring the DAS in a specific configuration to be an agent.

The *i\**-based technique [86] [16], however, uses the *domain* and *target system* abstraction [10], which offers a compelling divide-and-conquer approach to mitigating the problem of DAS complexity. The agent-orientation of *i\** as well as bringing with it the benefits discussed in the previous paragraph, also offers a convenient way of separating the concerns and behaviour of DAS adaptive infrastructure from the DAS's adaptive behaviour, and from the system's business logic. The substantial tool support for *i\** modelling in general is also beneficial, limiting the task of providing tool support to one of extension rather than creation. The limited formalism support in *i\** hinders model-derivation and run-time representation type activity, and existing work modelling DAS behaviour in *i\** is a little less mature than with KAOS.

The domain and target system paradigm used in [86], and first discussed in [10], combined with the agent orientation of *i\**, offers the possibility of better limiting modelling complexity than does the KAOS support, and it is this ability to limit modelling complexity that has underpinned the decision to extend the *i\**-based process to address this thesis's research questions.

Starting with the next chapter, this thesis presents its contributions, starting with a detailed look at the extended [86] process, better supporting ease-of-change through improved traceability, and discussion of how the models can be used to derive adaptive behaviour, and how a DAS can achieve greater autonomy by itself using the models.

## 4 ReAssuRE Modelling for DASs

---

The ReAssuRE modelling process centres around the recording of assumptions made during the requirements modelling process, as eluded to by the full name: Recording of Assumptions in RE. The LoREM [16] modelling process that the work extends, and i\* in general provides no means of recording the decisions made during the modelling process, which is to some extent supported in KAOS [70]. By recording assumptions on DAS models, it becomes possible to revisit them in light of changed information or new understanding as necessary. This revisiting of previously made decisions can be by humans later in the RE process, or potentially by the DAS itself at runtime.

This chapter details the ReAssuRE modelling process, focussing on the improvements made over LoREM [16], and how the improvements offer the possibility of a DAS operating with a greater degree of autonomy. The chapter discusses modelling at Levels One and Two of the levels of RE for DASs identified in [10], with these first two levels covering the adaptive behaviour of DASs. The chapter discusses benefits, drawbacks and limitations before drawing its conclusion.

### 4.1 The LoREM Process

The ReAssuRE process is based on the existing LoREM modelling method [16]. To aid in understanding how ReAssuRE models are created, used and how the ReAssuRE process itself fits in with a wider requirements engineering program, this section describes and discusses the underlying LoREM process.

The environment in which a DAS-to-be is to operate is partitioned into distinct domains. This partitioning is no easy task, with individual domains needing to be foreseen, with identifiable characteristics found to enable the DAS to identify the prevailing domain in all circumstances. Errors made during this process are amongst the most difficult to

rectify, requiring potentially multiple new target systems to be devised, or if border conditions were incorrectly set, multiple target system designs to be revised.

Suitable environmental partitioning is characterised by a finite number of domains being identified, each with monitorable conditions attached. The detection of an individual domain's monitorable condition(s) signalling its prevalence. Furthermore, an understanding of the way the environment will change between domains is required: can some domains only be arrived at before or after another? At what measured value can it be said the environment has changed from one domain to another? The results of this domain partitioning are not recorded explicitly in model form, although it could conceivably be beneficial to record them in a state diagram, for example.

LoREM modelling is not appropriate for systems and operating environments for which this is impossible, and once this partitioning has been achieved the LoREM process dictates the modelling of a target system for each identified domain. This modelling features little variability, with an individual target system specified to operate in a single steady domain. Decisions as to which individual configuration options, either the selection of components or their configuration, are used for an individual target system are taken before the overall configuration is recorded in a model. This modelling activity is referred to as level-one modelling.

Level-two modelling concerns the transitions between specified target systems, specifying the monitored condition that triggers the need to adapt, as well as the reconfigurations performed as part of the adaptation. This modelling activity relies on the border conditions identified during partitioning, and on the specification of individual target systems recorded during Level One modelling.

Level three modelling is essentially modelling of the requirements for an adaptation infrastructure to allow the environmental monitoring and target system transitions described by the Level-Two models to be performed. This modelling activity relies on knowledge of the monitoring and reconfiguration capabilities described in the Level Two models.

Although the literature on which LoREM is based [10] describes a fourth level of RE performed for DASs, this covers research on adaptation mechanisms in general. Modelling this type of abstract, generalised RE activity is of little value to an individual DAS, and thus no modelling activity is associated with this Level Four RE in LoREM.

To conclude, LoREM can best be thought of as a series of modelling perspectives from which a DAS-to-be is viewed. Although the reference to individual perspectives as “levels” could be thought to imply an increasing level of abstraction, this is not the case. Level-two modelling explores the requirements for transitioning between the target systems modelled at Level One. Level-three modelling, in turn, explores the requirements posed by the activity modelled at Level Two. The ReAssuRE modelling process improves upon LoREM at Level One and two to provide richer tracing information which can be used later in the requirements and software engineering processes, or by the DAS itself at run time. The improvements made at Level One and two are discussed, in turn, in the following sections.

## 4.2 Level One Modelling

The existing LoREM process [16] requires an *i\** Strategic Dependency (SD) model to be created for the DAS as a whole. Additionally, an *i\** Strategic Rationale (SR) model is created for each of a DAS's target systems, showing the configuration adopted in each domain, how the prescribed configuration satisfies the DAS's goals, and the extent to which the configuration satisfies the DAS's softgoals. From a tracing perspective, this provides only limited information in the event that the decisions need to be re-examined later in the RE process. Figure 2 and Figure 3 show a Level-One LoREM SD model of an adaptive image viewer, and an SR model of an individual target system, respectively. The adaptive image viewer was first described in [14].

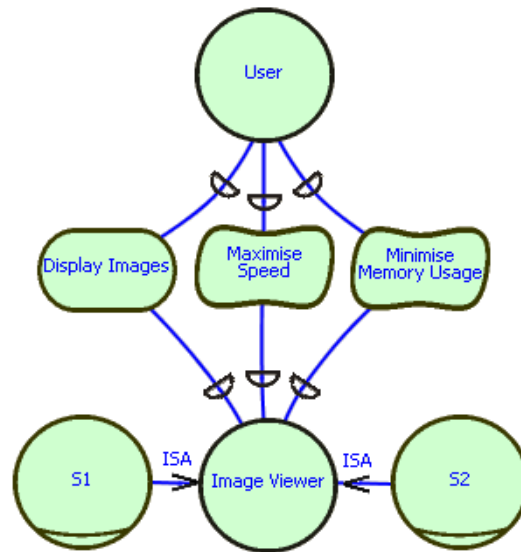


Figure 2: Level 1 LoREM SD Model for Adaptive Image Viewer

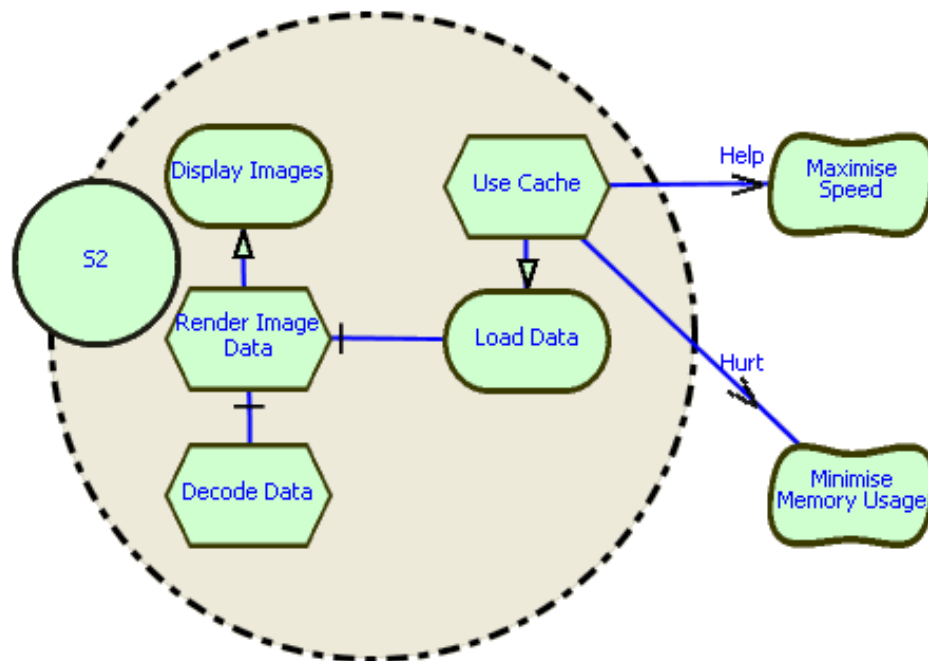


Figure 3: Level 1 LoREM SR Model for Adaptive Image Viewer's  $S_2$  Target System

The adaptive image viewer is used here, and throughout the chapter as a simple DAS to illustrate the concepts being discussed. The adaptive image viewer is an application

designed to display images to a user, loaded from a networked file store. The adaptive image viewer uses adaptation to balance two conflicting concerns: the desire to minimise memory usage, and the desire to maximise the speed at which images are loaded and displayed. The DAS's only adaptive capability is to enable or disable a file cache, which pre-loads images in an attempt to improve image loading speed, at the expense of significantly higher memory consumption.

The Level-One SD model depicted in Figure 2 shows a high level view of the actors, goals and dependencies in the adaptive image viewer. An overview of the syntax of SD models is included on page 38. In this simple DAS, a user depends on the adaptive image viewer to display images, and to do so as quickly as possible whilst using as little memory as possible.

Figure 2 also depicts the DAS's two target systems, which reflects the DAS's operating environment having been partitioned into two domains: low latency (referred to as  $D_1$ ) and high latency (referred to as  $D_2$ ). The low latency domain is characterised by good network performance, which means that the delay encountered when loading images will be negligible. Conversely, the high latency domain is characterised by poor network performance, which introduces a noticeable delay. For each of these domains, a separate target system is devised, specified and created. The  $S_1$  target system is tailored to the  $D_1$  domain, and  $S_2$  for  $D_2$ .

Figure 3 shows the LoREM Level-One model for the  $S_2$  target system, which is intended to operate in the  $D_2$  domain. The DAS's goal is to “Display Images”, which can be satisfied by completing the task: “Render Image Data”. The “Render Image Data” task can be decomposed into: “Decode Data” and “Load Data”. In the  $S_2$  target system, the “Load Data” goal is achieved by completing the “Use Cache” task, indicating the use of the file cache in domain  $D_2$ .

Given the likelihood of DAS' requirements models needing to be changed throughout the RE process, as a result of uncertainty in and about the operating environment, ReAssuRE seeks to offer improved traceability in DAS requirements models. Looking at

Figure 3, it is possible to infer that a decision has been taken to require the adaptive image viewer to use a cache when operating in the  $D_2$  domain, and that using the cache improves the perceived speed of the adaptive image viewer whilst increasing memory usage. However, it isn't necessarily possible to revisit the decision at a later date: there's no record of the alternatives considered, or what their expected impacts on the softgoals would be. In order to offer this kind of backwards traceability, it will be necessary to record several things on the Level-One model:

1. The alternative goal satisfaction strategies considered
2. Their impact on the DAS's softgoals
3. The decision reached
4. The rationale behind the decision

Recording the alternative goal satisfaction strategies considered is merely a matter of adding additional tasks representing the rejected alternatives to the model. The rejected alternatives' impacts on the DAS's softgoals can be captured in the same way as the impacts currently recorded: using standard  $i^*$  contribution links. The reached decision can be highlighted visually on the task representing the selected alternative. Recording the rationale behind the decision, however, is a somewhat trickier task. This thesis presents the idea of using *claims* to record this decision rationale, and it is this concept that is discussed in the next section.

### 4.2.1 Recording Rationale with Claims

One of this thesis' contributions is the addition of *claims* to  $i^*$  models as a means to record the rationale behind a decision as well as its outcome, along with recording the alternatives considered. The availability of this information improves traceability and

allows decisions to be re-taken later in the RE process, or possibly even by a DAS itself at run time. Claims are not a new concept: they formed part of the NFR framework [89], on which  $i^*$  is itself based.

In the NFR framework [89], claims could be used for two purposes: to argue the rationale behind an individual decision, or to support a softgoal prioritisation that can affect multiple decisions. The second use, softgoal prioritisation, has an equivalent in the related GRL ontology [98], but it is the first type of use that this thesis advocates for DASs. The use of claims to improve traceability in DAS models has been published in [87].

There is a related concept in the  $i^*$  ontology, called a *belief*. A belief is something an actor in the system holds to be at least partially true, and like a claim it can be used to affect a softgoal or contribution link. The principal difference between a claim and a belief is that a claim is presumed true by the analyst whilst a belief is not. There is also a syntactic difference, in that a Task may exert influence over a belief via a contribution link, (or rather, the degree to which an actor holds the belief to be true).

Claims are used in two different types of model during the ReAssuRE process: in the Level-One SR models created for each of a DAS's target systems, and in so-called *Claim Refinement Models* (as introduced by the NFR framework [89]) to examine the basis of assumptions. Returning to the Adaptive Image Viewer example discussed earlier in the chapter, Figure 4 shows the Level-One SR model for the same  $S_2$  target system that was depicted in Figure 3 on page 50. Unlike Figure 3, Figure 4 shows both of the alternative satisfaction strategies for the “Load Data” goal considered, shows which strategy was selected (“Use Cache”) and uses a claim to record a brief representation of the rationale behind the decision.



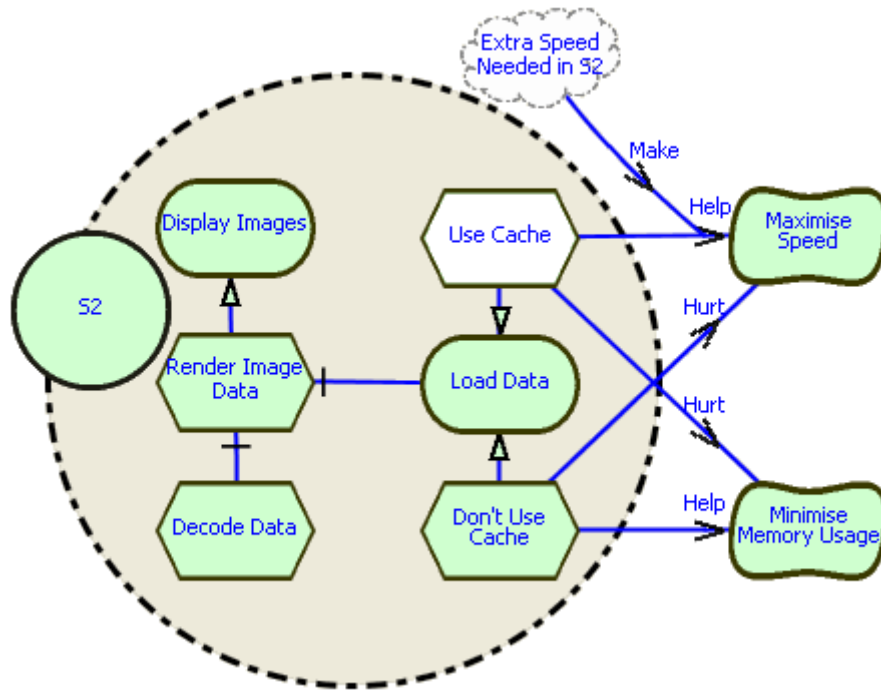


Figure 4: ReAssuRE Model of Image Viewer's  $S_2$  Target System

The claim “Extra Speed Needed In  $S_2$ ”, represented by a cloud-like shape in Figure 4, uses a *make* contribution link, to assert the importance of the *help* contribution link running from the “Use Cache” task to the 'Maximise Speed' softgoal. Given the importance of this positive contribution, the “Use Cache” goal satisfaction strategy is selected over the “Don't use Cache” strategy. The claim breaks the apparent deadlock in the contribution links on the model, asserting that the extra speed gained by using the cache is of greater importance than the memory saved in its absence.

The same rationale can be similarly represented using a claim that *breaks* a contribution link on the model, as depicted in Figure 5. The contribution link used affects the phrasing of the claim needed to convey the decision rationale, and in some cases the accuracy and ease-of-understanding of the rationale recorded will differ between notations. In this instance, the rationale expressed by the claim in Figure 4 is clearer.

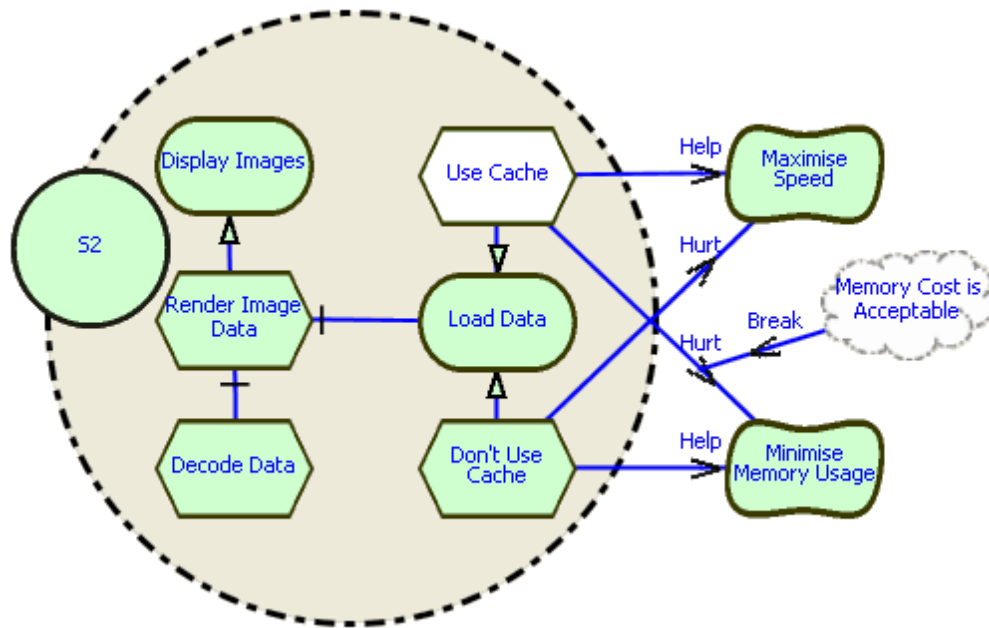


Figure 5: Claim using a “break” Contribution Link

In this example, there is little difference between the two notations. However, the type of contribution link a claim uses, and the polarity of the contribution link to which the claim is attached can affect the claim's meaning on more complex models. Some combinations of claims' contribution links and the links to which they are attached can be described as *inclusive* and some *exclusive*, as represented on the table below:

Claim Contribution Type	Attached Link Polarity	Description
Make	Positive	Inclusive
Make	Negative	Exclusive
Break	Positive	Exclusive
Break	Negative	Inclusive

Table 1: Claim Contribution and Attached Link Inclusiveness

The combinations in Table 1 described as “Inclusive” have the effect of making the task the “attached” contribution link relates to more likely to be selected, and those

described as “Exclusive” lead to the task being wholly or partly removed from consideration. Inclusive claims are used to argue that a positive contribution is important or necessary, or a negative contribution is insignificant. Exclusive claims argue a positive contribution is insignificant or unnecessary, or that a negative contribution poses an insurmountable problem. Using an inclusive notation allows the decision rationale to be recorded with one claim, whereas exclusive combinations will require a claim to exclude each rejected strategy. For decisions with more than two alternative goal satisfaction strategies, there is thus a preference for recording rationale using inclusive claims where feasible.

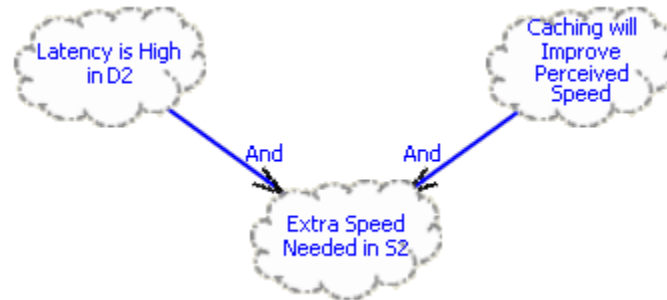
A claim is written as a simple statement of fact (in essence, the analyst is *claiming* something to be true), which means that a claim can be characterised as a record of an assumption being made. It is possible to examine the basis of a claim or an assumption using a *Claim Refinement Model*. The use of Claim Refinement Models promotes assumptions as first class entities à la KAOS, offering broadly similar reasoning capability. These models are discussed in the next section.

## 4.2.2 Examining Assumptions with Claim Refinement Models

Claim Refinement Models were introduced in the NFR framework [89], and use a tree-like structure to connect a claim under study with *underlying* claims (sometimes called sub-claims) which represent the basis of the original, studied claim. Claim Refinement Models are created on a one-per-target system basis in the ReAssuRE process, with a single model allowing the (potentially interdependent) bases of all the claims on a target system's Level-One SR model to be examined together.

A Claim Refinement Model serves as a record of the reasoning behind the claims on a target system's Level-One SR model. Broad, high-level assumptions are shown on the Level-One Claim Refinement Model and are broken down into smaller, more specific assumptions and assertions about the environment or a decision alternative. These

assumptions and assertions are also modelled as claims on the diagram. Figure 6 shows the claim refinement model for the Adaptive Image Viewer's  $S_2$  target system.



*Figure 6: Claim Refinement Model for Adaptive Image Viewer's  $S_2$  Target System*

Figure 6 shows that the “Extra Speed Needed in  $S_2$ ” claim, which supported the decision to use cache in the target system's Level-One SR model in the previous section (Figure 4), is derived from the conjunction of two underlying claims: “Latency is High in  $S_2$ ” and “Caching will Improve Perceived Speed”. The “Latency is High in  $S_2$ ” claim is an assertion on the environment, and given that significant latency is  $D_2$ 's defining characteristic, this is a relatively safe assertion. On the other hand, “Caching will Improve Perceived Speed” is an assumption about the performance of the file cache. This assumption can be considered less safe, given that there may be some patterns of file access the cache cannot predict (non-sequential viewing, for example).

The need for both underlying claims to hold in order for the “Extra Speed Needed in  $S_2$ ” to hold is signified by the *and* relationship. Other valid relationships between claims on a claim refinement model are *or*, *make* and *break*. A claim backed by two or more claims connected with *or* relationships will hold if any of the underlying claims hold. A claim single-handedly upholds another using a *make* relationship, and single-handedly invalidates another using the *break* relationship, although the last two relationships are less often used.

### 4.2.3 Benefits and Limitations

It is possible to use fine-grained contribution links to replicate the use of a claim in a Level-One model in some circumstances, although to do so is essentially a misuse of *i\** contribution links. The strength of a contribution link is intended to speak to the magnitude of an element's impact on a softgoal, whereas a claim is used to speak to the importance of an impact. The relatively small number of available fine-grained contribution links (7: 3 positive, 3 negative and 1 neutral) means that some decisions and associated rationale cannot be captured through the misappropriation of fine-grained contribution links, as will be discussed shortly. Furthermore, if the models already (legitimately) use fine-grained contribution links to capture impact magnitudes, it becomes impossible to distinguish between “large” and “important” impacts.

Another method of recording decisions taken would involve assigning the softgoals on a model a relative priority. For example, Amyot & Mussbacher [98] assign softgoals numerical weights, and contributions to heavily weighted softgoals are lent additional credence. There are two problems with this approach: firstly, it is very difficult to devise a representative numerical scheme that captures the relative priorities of softgoals and contribution links, and attempting to do so risks offering the illusion of quantitative backing for what have, in reality, been subjective decisions. Secondly, there are some combinations of decisions that cannot be represented in this manner. Figure 7 shows a snippet of an *i\** SR model with two decisions affecting the same two softgoals.

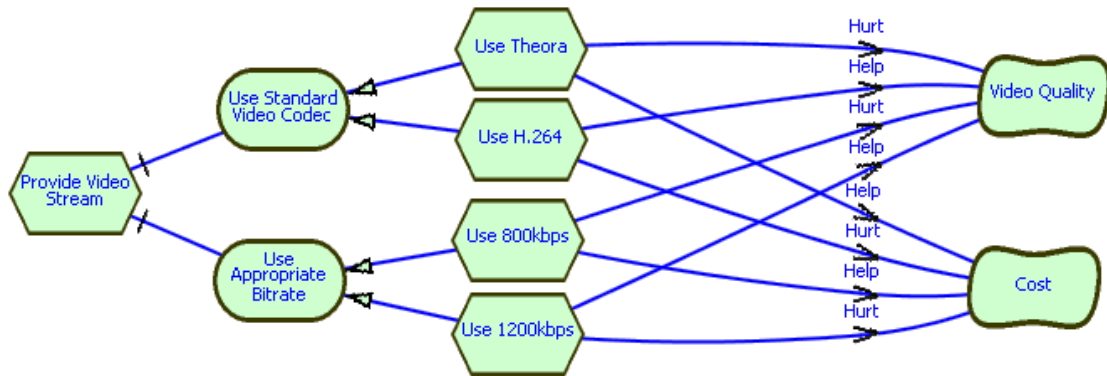


Figure 7: Partial  $i^*$  SR Model for Video Streaming System

Figure 7 shows two decisions taken when specifying an online video streaming system designed to allow viewers to watch a live sporting event. The two decisions concern the same two softgoals: “Video Quality” and “Cost”. The analyst aimed to specify a system that offers viewers the best possible video quality, but was working with a restricted budget. The budget for the entire video streaming system was fixed, but the analyst had freedom to assign budget between the video streaming server, the client software, bandwidth and web design freely, which means that at the decision level, cost is a relatively ill-defined softgoal.

The goal in Figure 7 named “Use Standard Video Codec” can be satisfied either by the task “Use Theora” or “Use H.264”, with Theora [99] being free to use and the commercial H.264 video encoder [100] offering better video quality. The goal “Offer Appropriate Bitrate” can be satisfied either by the Task “Stream at 800kbps” or “Stream at 1200kbps”, with there being a fairly linear relationship between the stream bitrate and bandwidth costs. It was calculated that the cost of the commercial H.264 video encoder was roughly equivalent to the cost of the additional bandwidth required for the 1200kbps video stream for the expected number of viewers. The relative improvement in video quality offered either by using H.264 or a higher bitrate was uncertain, so considered equivalent also.

The eventual decision taken for the video streaming system was to use the free Ogg Theora [99] video codec over the commercial H.264 encoder [100], but to offer the more costly 1200kbps video to improve video quality.

Assigning different weights to the “Cost” and “Video Quality” softgoals depicted in Figure 7 offers no way to reconcile the seemingly contradictory outcomes of the two decisions. In the more general case, any approach that records decisions based on some notion of softgoal priority will fail when handling multiple related, inconsistent decisions. These inconsistent decisions are often made when dealing with a budgeted requirement, which has a fixed satisfaction criteria across the system as a whole, but remains ill-defined at the level of individual decisions. For example, systems with a maximum total weight, cost or power consumption could all need to record decisions inconsistent with simple softgoal priorities, given the need to stick to an overall budget.

The ReAssuRE method of recording decision rationale, using claims on a per-decision basis allows inconsistent or directly conflicting decisions to be recorded, along with the rationale behind them. The rationale of inconsistent or conflicting decisions is particularly useful given its complexity, and particularly difficult to infer from models if not stated explicitly. The same  $i^*$  SR model augmented with claims allows the decisions to be recorded, as depicted in Figure 8.

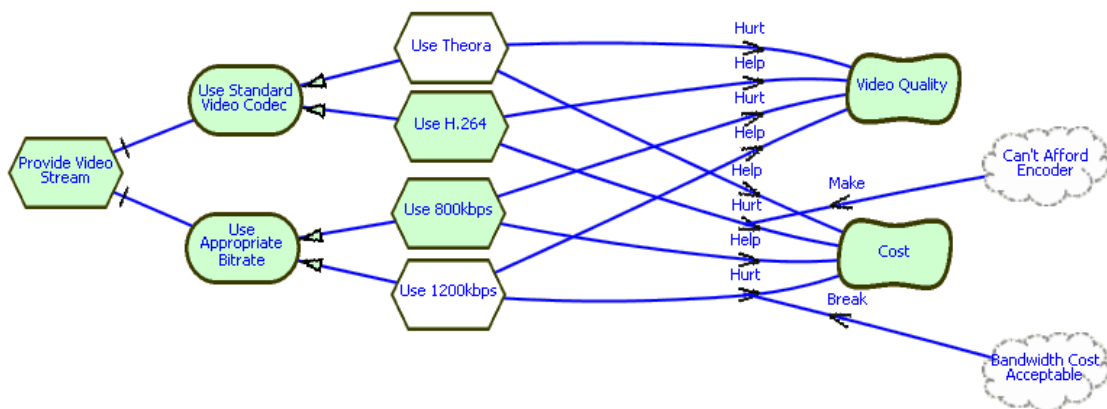


Figure 8: Claim-Augmented  $i^*$  SR Model of Video Streaming System

Figure 8 shows the decisions reached in each case, which cannot be achieved by manipulating softgoal priorities. The claims also give a brief overview of the rationale behind each decisions, which in this case are clearly inconsistent. The crucial factor in these particular decisions was the possibility of costs being lower should fewer people view the event, and the full rationale behind the two decisions is depicted in a Claim Refinement Model in Figure 9.

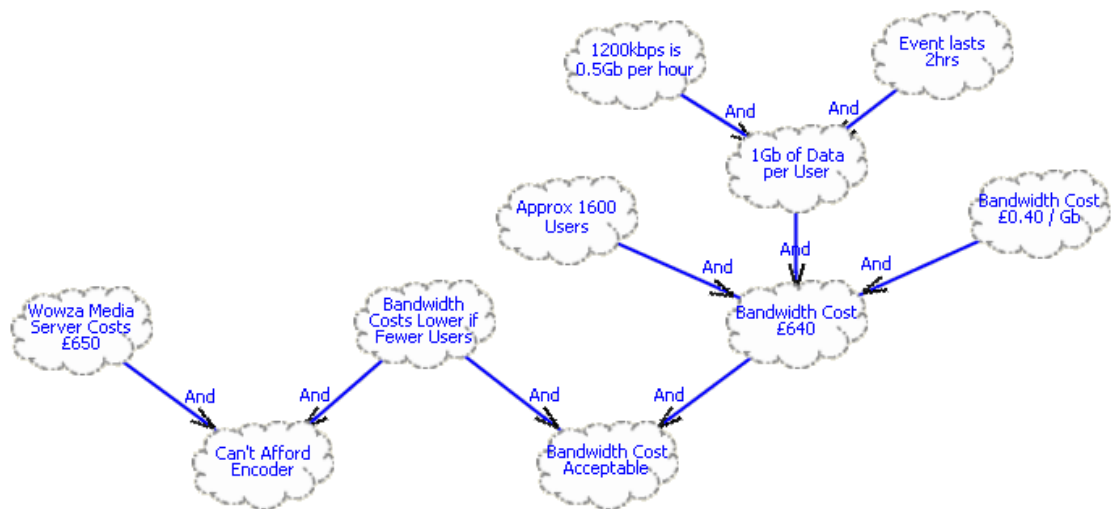


Figure 9: Claim Refinement Model for Codec & Bitrate for Video Streaming System

Figure 9 shows the cost of the software necessary to provide H.264 video and the working behind the bandwidth cost estimate. The model serves to explicitly highlight the estimates of event duration and number of users, and to show the relatively perilous nature of the decisions, given their reliance on estimates. The key assertion in the model, however, is the claim stating that bandwidth cost [unlike the H.264 software cost] will fall if there are fewer users than expected, which as stated previously underpinned the decision to use the higher bitrate stream instead of better quality encoding for the video.

The greatest benefit of using claims to record decision rationale in Level-One models is the enhanced traceability of the models with the claims added. Enhanced traceability is useful when making changes to the models. There are many potential reasons that a DAS's model may need changing: environmental knowledge may improve (or merely change), a



recorded assumption may be proven false, a new potential goal satisfaction strategy may be identified, a user requirements change, or even a need to make a change as a result of a previous change. With a complex, volatile and possibly imperfectly understood operating environment, these change drivers are likely to be encountered as least as frequently when performing RE for DASs as when for traditional, non-adaptive systems.

Regardless of the reason for a model change becoming necessary, the ability to identify the decisions affected by a change is crucial to an efficient change-management process. The enhanced traceability of ReAssuRE models over LoREM models allows decisions affected by a change to be identified, re-considered, and re-taken if necessary, which is clearly useful.

As well as the ability to identify decisions and trace change impact, ReAssuRE models allow the problem space to be explored by way of uncovering assumptions. The act of constructing a Claim Refinement Model examining one or more claims forces the analyst to think about the assumptions they have made in making decisions, and provides an opportunity to validate newly uncovered (previously implicit) assumptions. The ability to uncover previously implicit but now validatable assumptions can, potentially increase the robustness of the eventual DAS's specification. For those assumptions for which validation is impossible or infeasible, another of this thesis's contributions allows assumptions to be monitored at run time, and for decisions reliant upon them to be re-taken by a DAS autonomously in the event of one or more assumptions no longer holding. This contribution is discussed further in Section 6.2.

Although the benefit of modelling decisions and assumptions with claims in terms of traceability is applicable to the modelling of any system, this thesis restricts itself to DASs for two reasons: Firstly, the complex, volatile and uncertain environments that DASs prove most useful for mean that there is a greater probability of decisions having to be re-examined or re-taken later during the RE process. Secondly, decisions taken during the RE process for “partitioned” DASs, designed to operate in an environment that has been partitioned into individual domains, are likely to be repeated in multiple domains. As such, the rationale behind the decision in an individual domain is more likely to become

lost or confused, given the presence of near-identical decisions taken in multiple other domains.

The idea of recording key decisions and exploring the assumptions on which they are based is shared with the notion of obstacle analysis in KAOS. However, the approaches differ in style, application and their suitability for DAS modelling. Obstacle analysis involves uncovering potential problems, threats or conditions that may block or impede the satisfaction of a goal. After uncovering a potential obstacle, an addition can be made to the model to mitigate the obstacle. The additions to the model eventually form part of the system's specification, increasing its robustness.

At its core, the act of examining potential threats to the system's ability to operate (using KAOS obstacle analysis) is similar to the act of examining the assumptions upon which the specification relies (using claim refinement): any assumption not holding can be thought of as an obstacle. There are two key differences, however, between obstacle analysis and claim refinement. Firstly, obstacle analysis operates at a lower level, requiring specific knowledge of a threat or condition that can prevent a goal being satisfied. When performing claim refinement, the reason for an assumption's no longer holding is not necessarily explored, which means that claim refinement can take place at an earlier stage of the RE process, and with less concrete knowledge of the operating environment. Secondly, the specificity of obstacle analysis allows an analyst to introduce specification elements designed to mitigate identified obstacles. The higher-level nature of claim refinement means that it is difficult to fully understand the causes of an assumption no longer holding, and thus difficult to specify a specific solution.

For a DAS, the ability to explore assumptions at a higher level without the need for specific, reliable and deep understanding of what is likely a complex, volatile operating environment is clearly an advantage. However, the difficulty in exploring mitigation is challenging. One of this thesis's contributions is a method by which the design of a DAS's target systems may be modified in response to an assumption no longer holding either at design time or at run time. This contribution is discussed further in Section 6.2.

Some argue that the complexity of  $i^*$  SR models hinders understanding, and ultimately the models' usefulness. Thus, adding additional elements to the models, in the form of claims, comes at a cost of additional complexity. Some  $i^*$  modelling tools [84] [73] offer the ability to view only relevant subsections of models, which goes some way to mitigating the problem. In the LoREM process [16], a significant number of models need constructing, which further increases the workload in their creation, a problem shared by the ReAssuRE process. Models constructed at a single of the four levels of RE for a DAS often have little in the way of variation, and in some ReAssuRE models, only the claims differ.

One potential method by which to address this issue would be to offer a method by which a single model could be constructed containing the model components applicable to all target systems (at Level One) with different “dimensions” conditionally displaying only the elements applicable to a specific target system. At its core, this idea is similar to the method proposed by Lapouchnian & Mylopolous [101] to offer contextual variability in goal models, with different target systems treated as contexts. This fits well, given that claims record an assumption made about the environment, a DAS component or an expected impact of a component's selection, and contexts in the work are defined as “abstraction[s] over a set of environmental assumptions”.

According to Lapouchnian & Mylopolous, model components (elements and links, which would naturally extend to claims) can be “tagged” to appear only (or not appear) in one or more certain named contexts. Switching between contexts brings a different dimension of the model into view. At present, however, no  $i^*$  modelling tools support multiple model dimensions, and their utility may be limited to high-variability models, of which DAS models can be argued to be subset.

Another potential problem with claim-augmented SR models is that it is perfectly possible to construct a model containing decisions for which the available alternatives are known, but for which it is impossible to infer the selected alternative. This problem can exist in any  $i^*$  or similar model, and is known as deadlock. Although it is of course possible to require all decisions to be supported by a claim breaking any deadlock, the possibility of

a model being modified by the DAS at run time (discussed further in Section 6.2) opens the possibility of the model becoming deadlocked. At this time, the only remedy to this issue is through careful construction of ReAssuRE models, and allowing only deterministic model modification to occur at runtime.

To conclude, the use of claims in DAS Level-One modelling is one of this thesis's contributions, and forms the foundation of several of this thesis's later contributions. Recording decision rationale, and the assumptions upon which they were based not only improves traceability, but opens up the possibility of a DAS performing model modifications at runtime in response to unanticipated environmental conditions, adopting new configurations based on the changed models. The next section examines the ReAssuRE improvements to the Level-Two modelling process.

### 4.3 Level Two Modelling

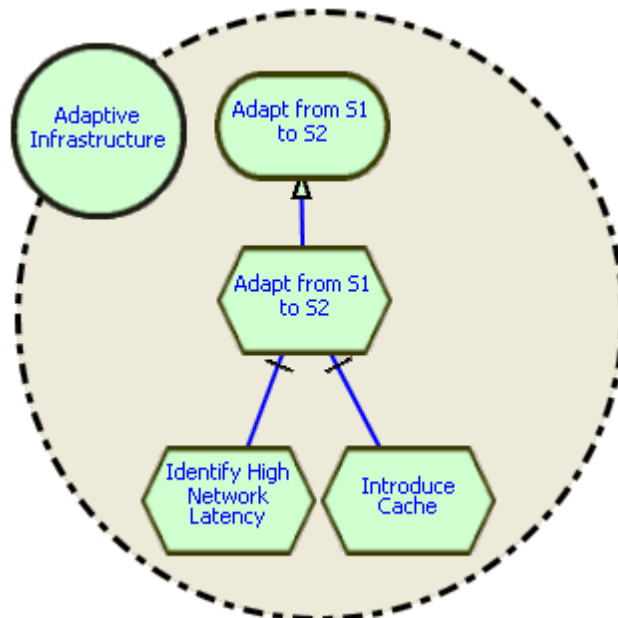
If individual target systems are modelled at Level One in the LoREM [16] process, then Level-Two models concern a DAS's transitions between them. Level-Two models cover the DAS's monitoring of its operating environment, the decision-making that uses this monitoring information to identify changes in domain, and the adaptation triggered in response to these domain changes, effecting the change to a new target system. Level-Two models allow the analyst to understand the DAS's adaptive behaviour, and taken with the Level-One models can offer a full picture of the DAS's behaviour in all expected conditions.

The Level-Two models used in the ReAssuRE process have been adapted from those used In the LoREM [16] process, aiming to make them more amenable to automated analysis. The requirements posed by the need to be amenable to automated analysis (and the analysis itself) is discussed further in Section 5.2. For this purpose, the Level-Two models perform two key functions:

1. Define valid transitions between target systems
2. Define the conditions under which those transitions will occur

Defining the valid transitions between target systems is merely a matter of creating Level-Two models for only pairs of target systems the DAS is to be able to transition between. Invalid transitions, that is, those pairs of target systems the DAS shall not transition between (or may do so only unidirectionally), are identified as such by there being no Level-Two model associated with them (or only a single model for unidirectional transitions).

The conditions under which a given adaptation is to occur is often referred to as a trigger condition: some property of the environment or the DAS whose detection causes the DAS to perform a given adaptation. Returning to the Adaptive Image Viewer example used earlier in the chapter, Figure 10 shows a LoREM Level Two model for the transition between target systems  $S_1$  and  $S_2$ .



*Figure 10: LoREM Level Two Model of Adaptive Image Viewer's  $S_1$ - $S_2$  Transition*

The transition depicted in Figure 10 is the transition performed when the Adaptive Image Viewer detects significant network latency, and introduces a file cache to improve perceived performance. The transition occurs between the  $S_1$  (no cache) and  $S_2$  (cache) target systems, as indicated by the “Adapt from  $S_1$  to  $S_2$ ” goal. The need to include a (identically named) task performing the adaptation is syntactic, prescribed by the  $i^*$  guidelines [102], and is completed by in turn completing two sub-tasks: “Identify High Network Latency” and “Introduce Cache”.

Earlier in the section, two key functions of a Level-Two model were identified. Examining Figure 10 with respect to these, the model clearly allows the valid transitions to be defined (by creating a model for each), but the circumstances in which the transition should occur are only imprecisely defined by the high-level “Identify High Network Latency” task. In order to better fulfil this second identified function, the Level-Two model needs to offer a lower-level view of the conditions under which adaptation occurs. The model also duplicates some information readily available on the Level One models: the need to introduce a cache to transition between  $S_1$  and  $S_2$ .

A lower-level view of the conditions under which an adaptation is to occur requires some representation of the conditions monitored by the DAS to trigger adaptation and what the trigger actually is. However, the monitored conditions and the adaptation trigger are the responsibility of two different elements of the adaptive infrastructure: the monitoring mechanism, and the decision-making mechanism. In Figure 10, these two roles along with the adaptation mechanism are conflated into the “Adaptive Infrastructure” actor, which plays all three roles in the adaptive image viewer. In some DASs, these roles are played by different agents, - possibly with little knowledge of one-another. Thus, it is advantageous to model the conditions monitored by the monitoring mechanism, the trigger fired by the decision making mechanism, and the transition effected by the adaptation mechanism. Figure 11 depicts a ReAssuRE Level Two model of the same transition with this increased detail.

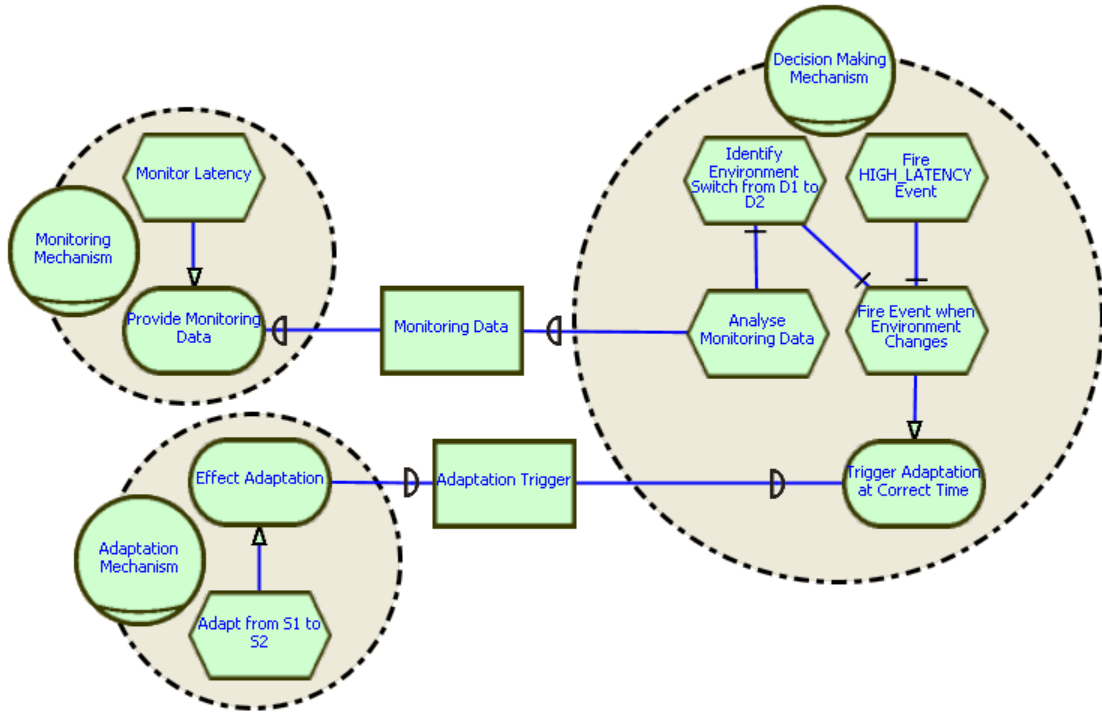


Figure 11: ReAssuRE Level Two Model of Image Viewer's  $S_1$ - $S_2$  Transition

The three roles depicted in Figure 11 each have a goal to achieve: the monitoring mechanism seeks to “provide monitoring data”, the decision making mechanism to “trigger adaptation at the correct time”, and the adaptation mechanism seeks to “effect adaptation”. These goals are the same for all transitions and for all DASs, with differences between transitions occurring in their means of satisfaction. The monitoring mechanism, in achieving the “provide monitoring data” goal furnishes the “monitoring data” resource, which is needed for the decision making mechanism to complete its “analyse monitoring data” task. The adaptation mechanism requires an “adaptation trigger” resource to be furnished to be able to adapt in a timely manner, with responsibility falling to the decision making mechanism.

The decision making mechanism is the most interesting part of the model in Figure 11, depicting how the goal “trigger adaptation at correct time” is achieved. In the Adaptive Image Viewer, adaptation is triggered by firing an event at the appropriate time, as

represented by the “fire event when environment changes” task. This task is completed by completing both the “identify environment switch from  $D_1$  to  $D_2$ ” task (which is in turn completed by completing the task “analyse monitoring data”), and the “fire HIGH\_LATENCY event” task. It is this final task which explicitly states the adaptation trigger required for the model to sufficiently define the conditions under which adaptation is to occur.

In some DASs, it would be beneficial to provide additional detail in the “analyse monitoring data” task, to explicitly state which monitoring values would indicate a need to fire the adaptation trigger. However, for the purposes of policy generation (as discussed in Section 5.2) this information is superfluous.

A ReAssuRE Level-Two model has the advantage of mapping more directly onto the three separate roles that make up an adaptive infrastructure than a LoREM Level-Two model. This allows the goals (and eventually requirements) of the individual roles to be examined separately, which is important in a DAS where the roles are to be (or merely may be) played by different concrete agents, or where the development of different aspects of adaptive infrastructure is likely to later be split between teams. ReAssuRE Level-Two models also provide more detail about the conditions under which a transition between target systems is to occur, which is necessary to support the derivation of a DAS's adaptive behaviour from models, as discussed further in Section 5.2.

These advantages, however, come at a price of complexity: in that ReAssuRE Level-Two models are considerably more complex than their LoREM counterparts. This adds to the modelling workload, and although this is of concern this thesis shows that the workload saved in allowing automatic derivation of adaptive behaviour, and the ability to trace errant adaptive behaviour back to its cause at the earliest goal modelling stages can justify this extra workload. It is also worth noting that most ReAssuRE Level Two models are similar (see for example Figure 12), offering the possibility of using templates of common elements to reduce this workload.



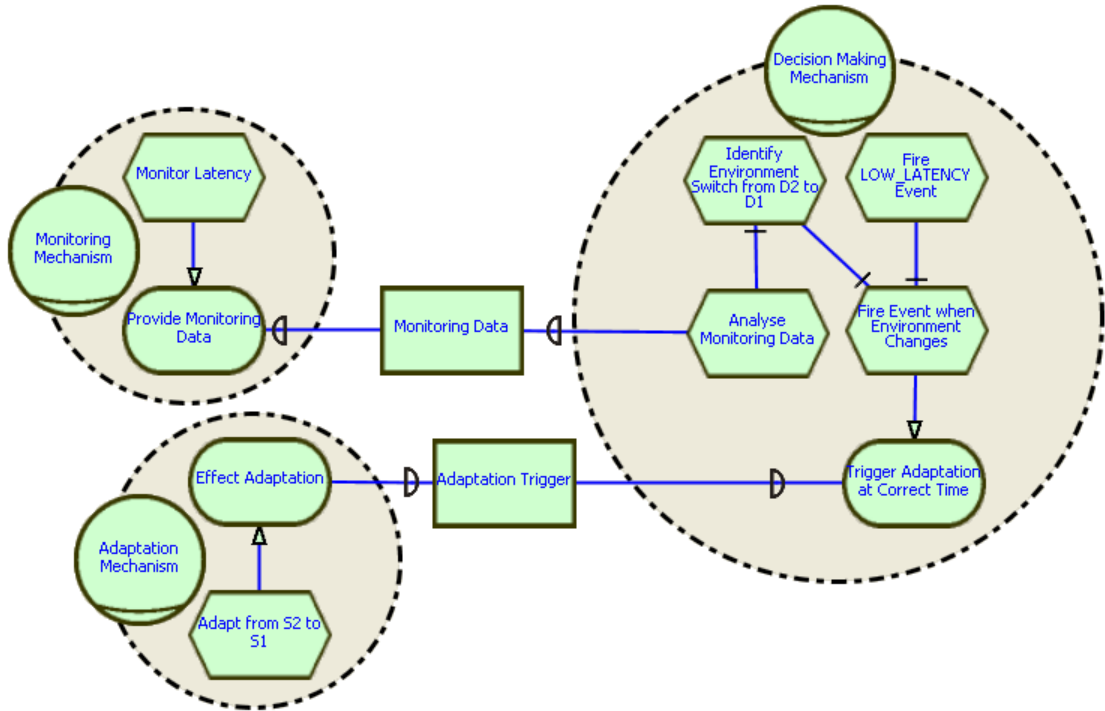


Figure 12: ReAssuRE Level Two Model of Image Viewer's  $S_2$ - $S_1$  Transition

The similarity between Figure 11 and Figure 12 also offers the possibility of the models being conflated, with differing elements displayed conditionally depending on the transition being viewed. This is essentially the same approach as discussed in Section 4.2.3, with individual transitions being considered as contexts [101].

To conclude, ReAssuRE Level Two models are constructed with greater detail, and are thus more complex, than their LoREM counterparts, which better enables automated reasoning based upon them. There is also significant advantage in this extra detail when dealing with DASs for which the three roles of an adaptive infrastructure (monitoring mechanism, decision making mechanism and adaptation mechanism) are to be separated.

## 4.4 Chapter Conclusion

This chapter presented and discussed the ReAssuRE modelling process for DASs, with a particular focus on changes made from the existing LoREM process. The chapter presented *claims*, which are used to highlight a decision in Level-One Strategic Rationale models, and to present the rationale behind it briefly. The chapter has also shown how the assumptions underpinning decision rationale can also be modelled as claims and examined in a *claim refinement model*. Finally, the chapter has shown how a DAS's transitions between target systems can be modelled at Level Two in such a way that adaptation triggers, which start a given transition can be identified for use later in the software engineering process, when writing adaptation policies.

The next chapter discusses how ReAssuRE models can be used later in the RE process aiding in managing requirement changes (demonstrating the enhanced traceability discussed in this chapter), in writing adaptation policies, and in performing requirements validation work.

## 5 ReAssuRE at Design Time

---

Chapter 4 introduces the ReAssuRE modelling process, using the adaptive image viewer example. This chapter extends the adaptive image viewer example to demonstrate the benefits of ReAssuRE models at design time. The chapter discusses the increased traceability of the models, providing examples of the sort of tracing activity the models support over and above their LoREM counterparts, along with their ability to support policy generation, and to aid in performing requirements validation.

The feature underpinning all of the model uses discussed in the chapter is the availability of extra tracing information, codified in claims in the models. Naturally, additional tracing information aids traceability, but the ability to retrieve some limited rationale aids also in identifying and exploring areas of uncertainty in the analyst's understanding of a DAS's proposed operating environment. Although it would be possible to perform the policy derivation discussed in Section 5.2 without information from claims (and thus it would be possible to perform the derivation from LoREM models), the process of identifying the proposed configuration for each of a DAS's target systems is simplified by the claims' presence.

### 5.1 Traceability in ReAssuRE models

This section examines both the drivers for, and the consequences of changes in a DAS's requirements models. The section is split into two along this divide, with the first subsection discussing the types of change a DAS model is likely to be subjected to and identifying the traceability requirements for ReAssuRE models to support change management effectively for each type. The second subsection demonstrates how the ReAssuRE models meet these traceability requirements, and support the tracing need. The issue of change in DAS requirements models, and the ability of ReAssuRE models to cope with this change has been published in preparation for this thesis [87].

### 5.1.1 Change in DAS Requirements

Changes can be required of a system at any stage during its design, implementation or useful life. In a system that possesses no adaptive capability, these changes are made offline by developers in the form of maintenance. A DAS, on the other hand, is able to perform some of these changes autonomously, lending DASs to use in volatile domains with shifting requirements and priorities.

The nature of the problem domains for which DASs are conceived are such that a DAS's operating environment may be only partially understood at design time. The novel and emergent nature of the technologies upon which DASs are based also means that the likelihood of beneficial new technologies emerging is high. A DAS may exhibit emergent behaviour as it re-configures itself dynamically, in ways and under circumstances that may have been hard to anticipate at design time. Thus, far from their adaptive capability making them immune to the need for maintenance and re-specification, DASs are particularly susceptible to it.

Ramesh and Jarke [103] sort users of trace information into two groups: high and low level, depending on the types of use made of the information. Two of these uses are of particular interest to DASs: 1. Tracing decision rationale and 2. Tracing for evolvability. To allow decision rationale tracing, a record needs to be made of each the viable alternatives considered, along with assumptions of the impact the selection of each would have. Requirements evolution can occur due to a change in user needs, or due to an identified deficiency in the system and requires the ability to understand how a requirement came to be.

As requirements at different levels of detail change, the question “Where did this requirement come from?” becomes harder to answer. Furthermore, given that these requirements may form the basis of the justification for several decisions, each of which will be repeated (with different environmental assumptions) for several target systems, a change in requirements can have a widespread impact in a DAS. The likelihood of far-

reaching change impact means that tracing of decision rationale and for evolvability are crucially important for DAS developers. Thus, DASs promote high-level tracing as essential practice.

There are five main classes of change that a DAS's specification may need to be adjusted for. Each needs handling differently.

1. Environmental Change
2. Faulty Assumption
3. New Technology
4. Consequential Change
5. Stakeholder Requirements Change

The following paragraphs discuss each identified class of change in detail.

Environmental change will be particularly common given the inherent volatility of any DAS's proposed environment; which increases the likelihood of individual domains being misunderstood, or entirely new domains being discovered. This class of change may occur during any stage of the DAS's life cycle, and the need for change may be signalled by a system failure if it emerges only after deployment. A change to the understanding of a previously identified environmental domain will trigger the need to re-evaluate the design decisions taken for it. The identification of a new domain will require a new target system (and associated model) to be created, possibly based on that of another target system.

When environmental understanding changes, the model for an entire target system will need to be reconsidered, or created from scratch if a new domain has been discovered. Essentially, this will involve re-making all the per-domain decisions (that is, those

decisions taken in each domain for each target system) in light of the changed (or new) environmental knowledge. For this type of change, the only trace information needed to effect the change efficiently is the ability to identify all the decisions taken for a given target system, although a record of the rationale behind the decisions taken will allow them to be re-taken more easily.

A faulty assumption may be an assumption about the environment in a given domain, or an assumption about a given component's suitability for a particular domain. The assumptions underpinning the Level-Two modelling are also classed as environmental assumptions, and will affect several different levels of model. Assumptions such as these may be broken as designers better understand the domain or the proposed DAS, or may only become apparent after deployment. An assumption being found faulty triggers the need to re-evaluate all decisions based upon it.

An assumption may be found faulty either by improved understanding of the environment or available alternatives for a decision, or having been broken demonstrably in the field. In either case, all of the decisions based on the faulty assumption will need to be revisited. In order to facilitate this forward tracing, there needs to be a record of all decisions reliant on the assumption, along with information on the alternatives previously considered for each affected decision (including their perceived impact on the DAS's quality features) to allow it to be re-taken.

The availability of an exploitable new technology, or more generally some newly identified capacity or opportunity for the DAS, can be modelled as a new alternative for some decision. Given the relative immaturity of adaptive infrastructures, this kind of change will likely occur frequently. As with a non-adaptive system, designers need to weigh the potential costs and benefits to decide whether to take advantage of the new technology or not. However, for a DAS, the decision needs taking in each domain, and reached for each target system. If the new technology is utilised in one or more target systems, other decisions that impact on those quality features affected by the use of the new technology in these target systems will need to be revisited, to provide an opportunity to re-balance them.

A new decision alternative will require only a limited number of decisions to be revisited. However, each decision may need to be revisited in each domain, for each target system. As such, to support this backwards tracing, there needs to be a record of alternatives previously considered for a decision, and a record of the rationale behind previously reached decision.

Consequential change so named because the change is necessitated as a consequence of a previous change. This kind of change will be particularly important in systems with a “budgeted” requirement such as a maximum power consumption or maximum total weight. In this case, making any of the previous types of change can require a re-evaluation of previous decisions across all domains, trying either to bring the system back down to budget if the change negatively impacted the budgeted requirement, or to more fully utilise the budget if the change created headroom.

A consequential change requires the ability to trace across target systems all decisions made that affected a given, budgeted requirement. As such, there needs to be a record of which requirements are affected by which selection alternatives, to allow the analyst to search through the system to find an acceptable change to trade-off against the previous, necessitating change.

Stakeholder requirements change is of course not specific to DASs. However, the impact may reach several target systems, essentially multiplying the workload involved in making the change. User requirement changes are difficult to predict, and are also variable in the extent of their impacts.

A stakeholder requirements change can potentially affect any or all target systems, or may necessitate a change to a static (i.e. identical in all target systems) component of the DAS. Therefore, as with user requirement changes to non-adaptive systems, the impact of this type of change varies from case to case.

To summarise, this section has argued that potentially incomplete understanding of a complex, volatile operating environment means that changes to a DAS's specification is likely. It has presented five classes of change that a DAS's specification may be subject to.

Each class of change presents its own tracing requirements for the change to be effected efficiently. Table 2 below summarises the traceability requirements for each class of change, and highlights those that are met by the LoREM and ReAssuRE models.

Class of Change	Traceability Requirements	LoREM Support	ReAssuRE Support
Environmental Change	Target system separation	Partial	Yes
Faulty Assumption	Forward tracing	No	Yes
New Technology	Backward tracing	Partial	Yes
Consequential Change	Backward tracing	Partial	Yes
Stakeholder Req. Change	Indeterminate	Indeterminate	Indeterminate

*Table 2: Support for Traceability Requirements of Identified Change Classes*

In Table 2, the target system separation tracing requirement refers to the ability to identify all the decisions taken for a target system, and the availability of the rationale behind the decisions. The forward tracing requirement refers to the ability to identify all the decisions *affected by* an entity, and the backward tracing requirement to the ability to identify decisions *that affect* an entity. The scope of a stakeholder requirement change is unpredictable, thus the tracing requirements for a specific change are also indeterminate.

The next section examines the degree to which each set of traceability requirements is met by the LoREM and ReAssuRE models, returning to the Adaptive Image Viewer DAS for illustration.

### 5.1.2 Tracing Examples

This section provides guided examples showing how ReAssuRE models meet the tracing requirements identified in the previous section. The examples all centre around the Adaptive Image Viewer first discussed in Section 4.2, and take the form of potential specification changes: one for each class of change identified in Section 5.1.1.



### Environmental Change

As understanding of a DAS's operating environment improves, the decisions reached and balances struck for a target system may need to be re-visited. Furthermore, even if the analyst's understanding of the operating environment is perfect, the environment itself may change over time, triggering the same need to re-visit a target system's specification. In Section 4.2, the Adaptive Image Viewer's two target systems were identified as:  $S_1$  (Normal) and  $S_2$  (High Latency). The DAS's specification calls for a file cache to be used in the  $S_2$  target system to improve perceived performance, but not for  $S_1$  to conserve memory. An example of an environmental change would be the emergence of a new domain:  $D_3$  - characterised by both poor network performance and low amount of available memory.

The emergence of this new domain would likely be identified by a failure during adaptation from  $S_1$  to  $S_2$ , when the DAS finds insufficient memory to enable the cache, prompting the need to specify a third target system:  $S_3$ . Given that the high-level goals and softgoals of the  $S_3$  target system are the same as the most similar existing target system,  $S_2$ , a significant amount of  $S_2$ 's model can be re-used for the new target system. The only model elements that should not be transferred are the claims, which are domain-specific. Figure 13 shows the newly created model.

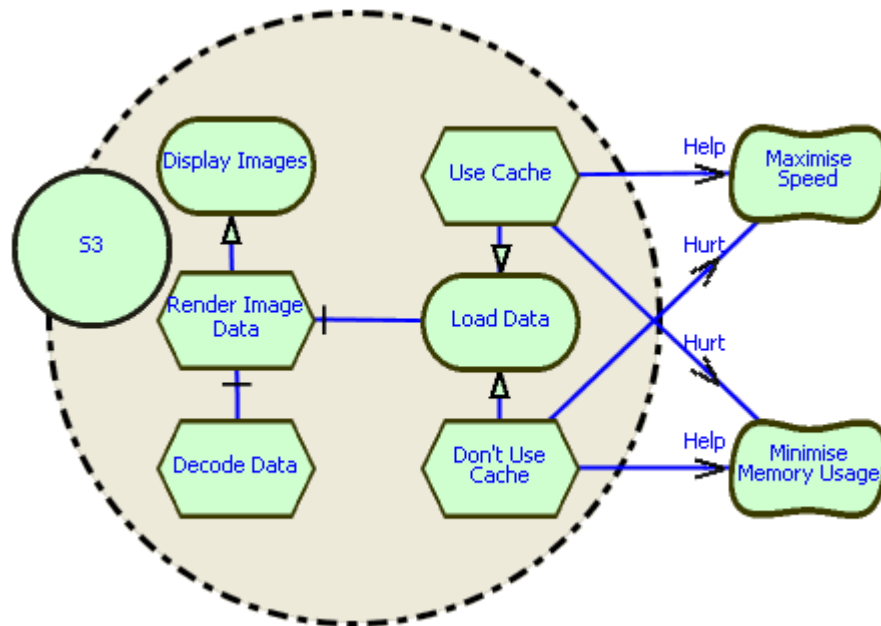


Figure 13: Transferred Elements of Adaptive Image Viewer's  $S_3$  Model

Now that a base model exists for  $S_3$ , the analyst can identify decisions that need to be made for the new target system. This identification is achieved by locating all goal elements on the model that have two or more tasks attached to it via means-end links. The only such goal in Figure 13 is the “Load Data” goal. The decision in this instance is simple: there is not enough memory to enable the cache, and so the “Don't use Cache” task is to be selected. It would be possible to model this decision by removing the “Use Cache” task from the model, but to do so would hinder future tracing by removing the record of available alternatives for the decision. Instead, the decision is to be modelled by the inclusion of a claim.

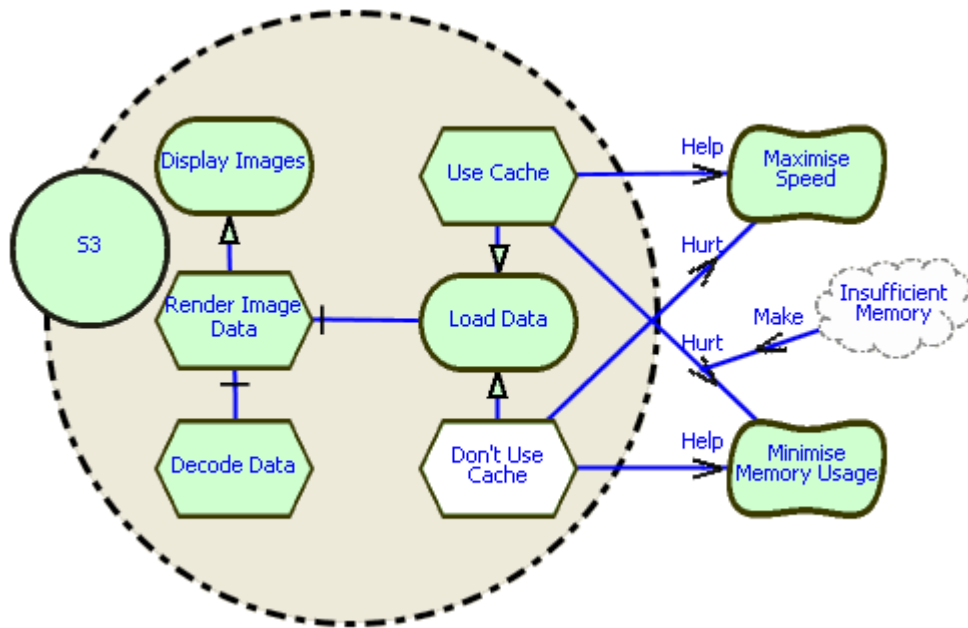
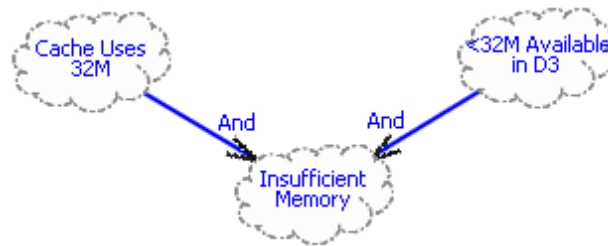


Figure 14: Final  $S_3$  Level One Model for Adaptive Image Viewer

The rationale behind the decision not to use the cache in  $S_3$  is that there is not enough memory to do so. Thus, the “Use Cache” task’s “hurt” contribution to the “Minimise Memory Usage” softgoal can be lent extra credence with a claim. The brief rationale in the claim: “Insufficient Memory” serves to assist later users of the model in understanding why the decision was reached, as well as serving as a bottom-level claim for a Claim Refinement Model. Figure 14 shows the final SR model for the  $S_3$  target system.

Figure 14 shows that the decision not to use a cache in  $D_3$  was taken, and that the rationale behind the decision was that there was “insufficient memory”. Although in this instance, even this brief rationale would likely be sufficient for tracing purposes, more complex rationales would be supported using a Claim Refinement Model. For illustrative purposes, the Claim Refinement Model supporting the “insufficient memory” claim is shown in Figure 15.



*Figure 15: Claim Refinement Model for Adaptive Image Viewer's  $S_3$  Target System*

Figure 15 shows that the “Insufficient Memory” claim is based upon two assumptions: that the cache uses 32M of memory when active, and that there is less than 32M of memory available in  $D_3$ . The former is an assumption about a decision alternative, the latter an assumption about the environment. Although the utility of claim refinement models to promote traceability is diminished for simple rationales, questions such as “Why does the adaptive image viewer not use a cache under these circumstances?” that may be asked later in the specification or design phases, or indeed after deployment are better answered with this information available.

### **Broken Assumption**

It could be argued that most types of specification changes are, in fact, underpinned by a perhaps implicit assumption that has been broken. However, in the context of this thesis, a broken assumption refers to one of the assumptions made explicitly during the RE process, modelled as a claim in a ReAssuRE model. Returning to the adaptive image viewer's  $S_2$  target system, the SR model for which was first depicted in Figure 4 on page 54, we can see that the  $S_2$  target system is specified as using the cache, because the extra perceived speed it brings was deemed necessary in  $D_2$ .

The  $S_2$  Claim Refinement Model (Figure 6 on page 57) shows that this rationale is based on two assumptions: that there will be significant latency in  $D_2$ , and that using the

cache will improve the perceived speed of the application in loading images. Unfortunately, this second assumption only holds if the cache is able to predict the images the user is likely to request next: e.g. if they are being accessed sequentially. If, in normal use, images are not accessed in a predictable sequence, the “Caching will Improve Perceived Speed” assumption will no longer hold.

ReAssuRE models support the tracing of the impact of a broken assumption by propagating a “broken” label through a Claim Refinement Model, and if necessary onto the SR model of a target system. The rules of label propagation throughout a Claim Refinement Model are as follows:

- When a claim is derived from a single claim, the derived claim inherits the label of the upper-level claim; the upper-level claim makes or breaks the derived claim.
- When a claim is derived from two or more AND-ed claims, it inherits the label of *any* of its supporting claims thus, should an upper-level claim be broken, so too is the derived claim.
- When a claim is derived from two or more OR-ed claims, a label is only propagated if *all* of the supporting claims possess the label.

Applying these rules to propagate a “broken” label throughout the claim refinement model in Figure 6 yields Figure 16.

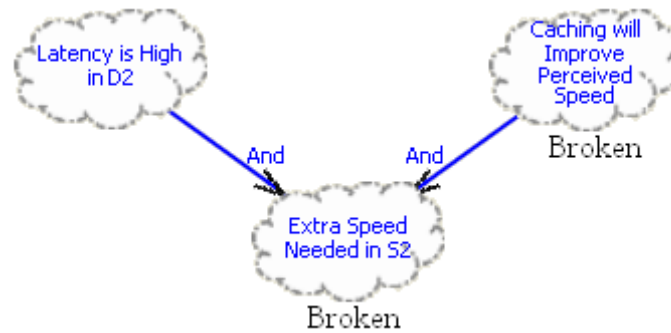


Figure 16: “Broken” Label Propagation in Claim Refinement Model

Figure 16 shows that if the “Caching will Improve Perceived Speed” claim is broken, so too is the bottom level claim: “Extra Speed Needed in  $S_2$ ”. This bottom-level claim is then broken on the  $S_2$  target system's SR model, where the decisions affected by the claim can be traced. The updated SR model for the  $S_2$  target system is depicted in Figure 17.

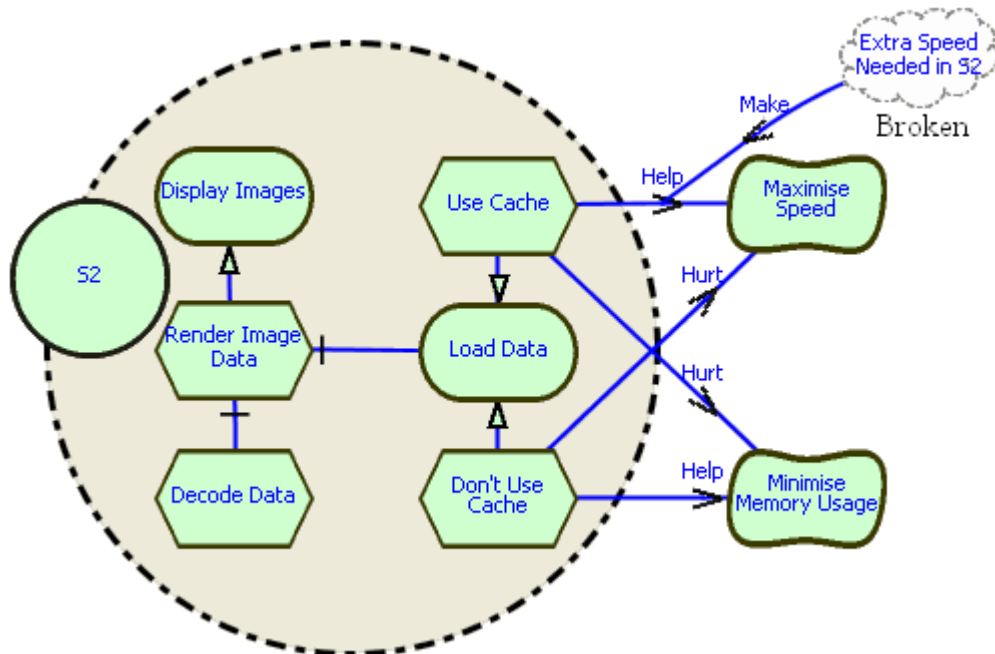


Figure 17: Adaptive Image Viewer's  $S_2$  Target System with Broken Assumption

As Figure 17 shows, the claim previously supporting the decision to use the file cache in  $S_2$  has been broken, and thus the contributions of the tasks requiring the use of the cache or otherwise are deadlocked once more. However, we can use the newly uncovered information: that the cache is ineffective in  $S_2$  to break the deadlock by adding a new claim: “Cache ineffective in  $S_2$ ”. Figure 18 shows the SR model for the  $S_2$  target system with the new claim.

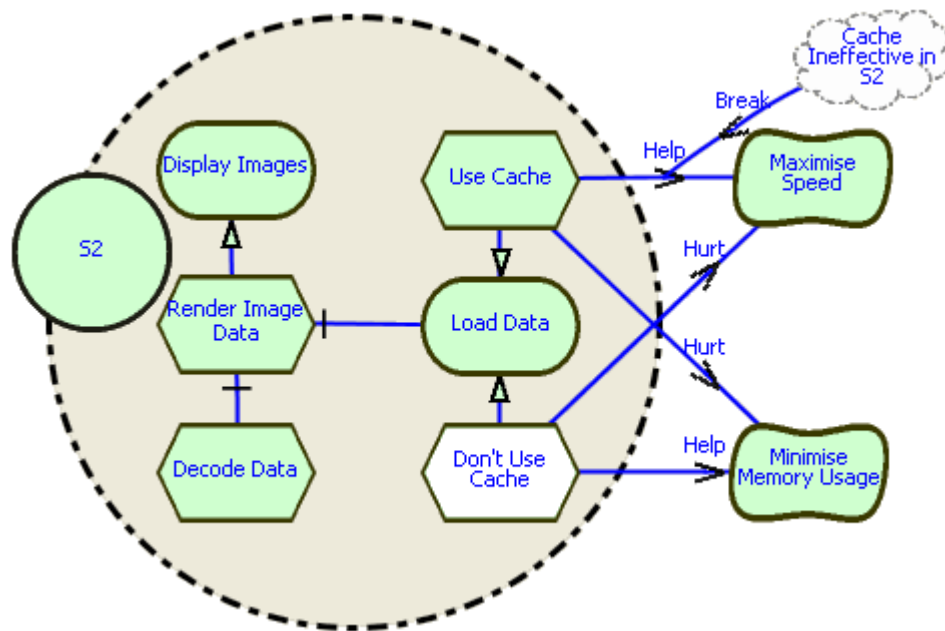


Figure 18: Adaptive Image Viewer's  $S_2$  Target System with New Claim

Figure 18 depicts the updated model with the faulty assumption's claim removed, and the new knowledge: that the cache does not, in fact, improve perceived speed in  $S_2$  added. If the newly-added claim (or rather, the rationale it eludes to) was complex, then it may be necessary to produce a new Claim Refinement Model showing the underlying assumptions. In this case, however, the rationale is simple and explicit: thus none is necessary.

### New Technology

A new technology will typically manifest itself as a new alternative to a decision, or a new decision will have to be taken because previously only one alternative was considered. For the adaptive image viewer, the example for broken assumptions can be extended to illustrate this type of scenario. If it was discovered that the image cache turned out to be ineffective, perhaps because the image viewer was used to view images non-sequentially, the idea of creating a new type of cache may be floated.

This new type of cache would use basic learning behaviour to remember the sequences that images *are* accessed in, so that if the same pattern of images is accessed again, the cache can pre-load image data and improve perceived speed. This new type of cache would need slightly more memory than the original cache to provide this learning feature as well as cache image data, but would opt not to cache image data when no prediction was available. Thus, the overall increase to memory consumption using the learning cache can be considered insignificant. Figure 19 shows an updated SR model for the  $S_2$  target system (with elements carried over from the previous broken assumption example) with the new type of cache added as a decision alternative.



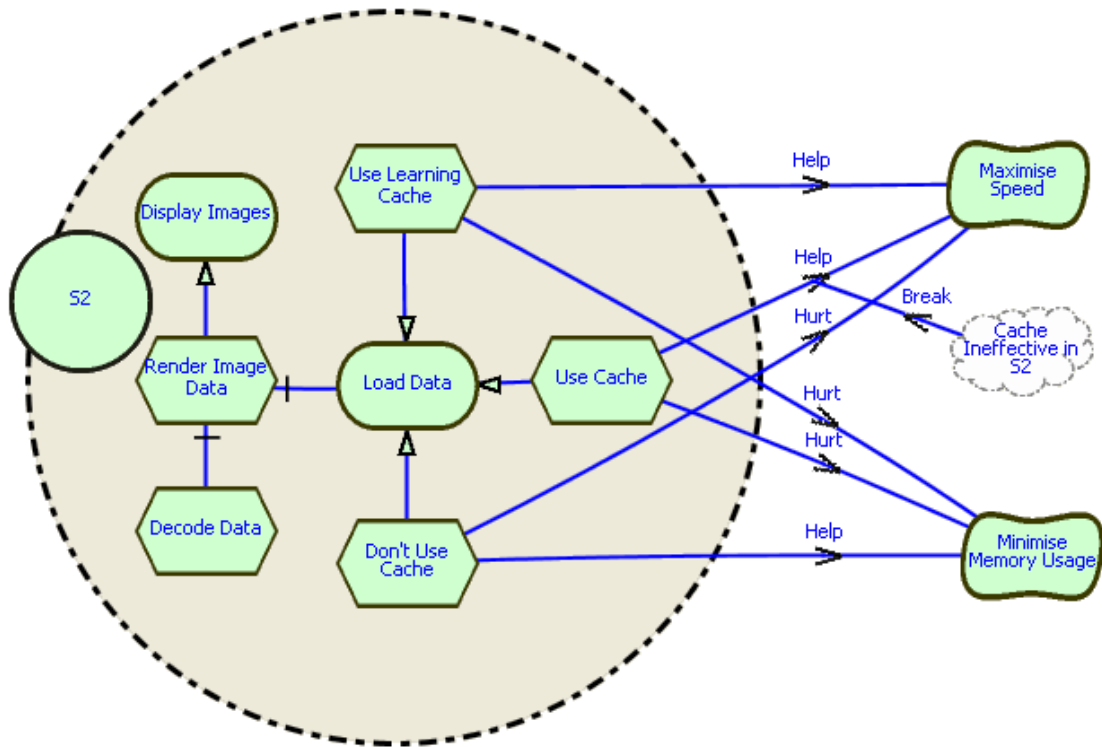


Figure 19: Adaptive Image Viewer's  $S_2$  Target System with New Cache Available

As Figure 19 depicts, using the original cache is still inadvisable because it has proven ineffective. However, given the use of an exclusive claim to remove the original cache from consideration, it is still viable to use either the new learning cache, or no cache at all. It is decided that, given the learning cache's relatively slim memory requirements in scenarios in which it will not be effective, that it is to be used in  $S_2$ . Using the learning cache in  $S_2$  will improve perceived speed at the expense of memory consumption where possible, but will use a small amount of memory for no benefit in less favourable scenarios. This rationale is somewhat more complex than that discussed previously, and the Claim Refinement Model capturing it is depicted in Figure 21.

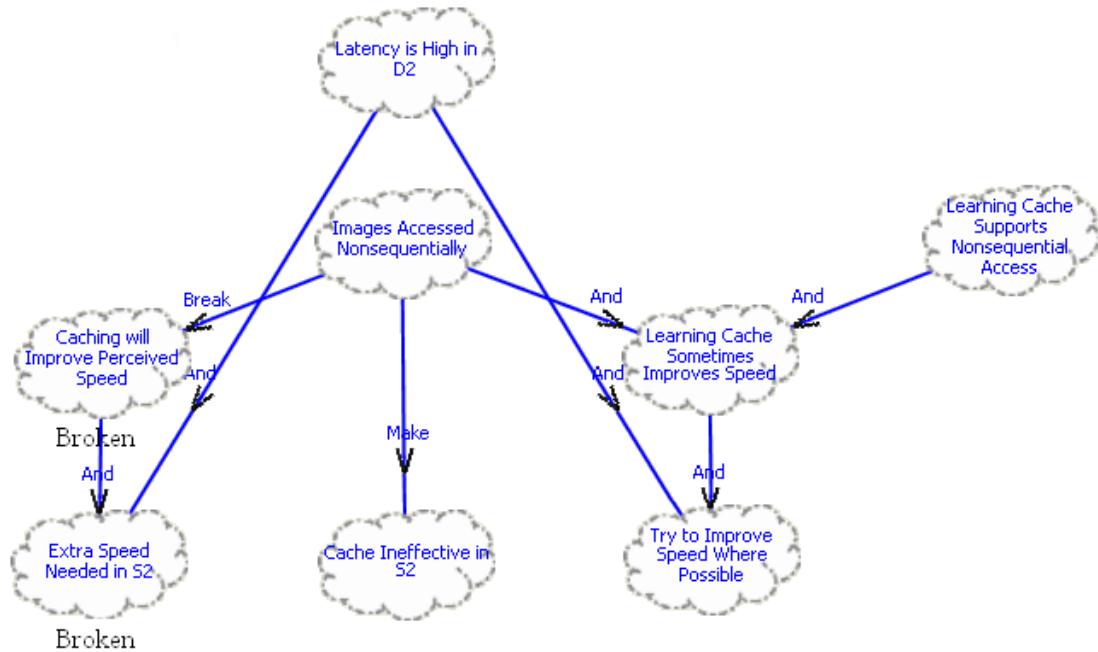


Figure 20: Claim Refinement Model with Rationale for Learning Cache in  $S_2$

The original cache failed because images were not normally accessed sequentially. this fact is modelled as a claim, which provides sufficient justification to support the “Cache Ineffective in  $S_2$ ” claim introduced in the previous example. The new cache's expected behaviour is asserted by the “Learning Cache Supports Non-sequential Access” claim, which conjoined with the previous claim supports the “Learning Cache Sometimes Improves Speed”. This, in turn, is combined with the previously seen “Network Latency is High in  $S_2$ ” assertion to offer a bottom-level “Try to Improve Speed Where Possible” Claim.

The original, and now invalidated, rationale persists in Figure 20. This is entirely optional, and offers the benefit of allowing previously made assumptions to be identified, even if later discarded. The ability to trace previous decision outcomes is not central to the ReAssURE process, but has some interesting uses, as discussed further in Section 6.2. Those who chose to remove invalidated decision rationale will enjoy less complex models, which is, of course, appealing.

Although the single additional bottom-level claim in Figure 20 would be sufficient to break the deadlock on the SR model in Figure 19, the rationale presented by Figure 20 makes no reference to the part of the rationale dealing with the acceptable memory cost in situations where the cache doesn't improve speed. This rationale is represented by the “Learning Cache uses Little Memory when Learning” claim, which along with the “Cache Ineffective in  $S_2$ ” claim, is responsible for upholding the bottom-level “Memory Cost Acceptable” claim, as depicted in Figure 21.

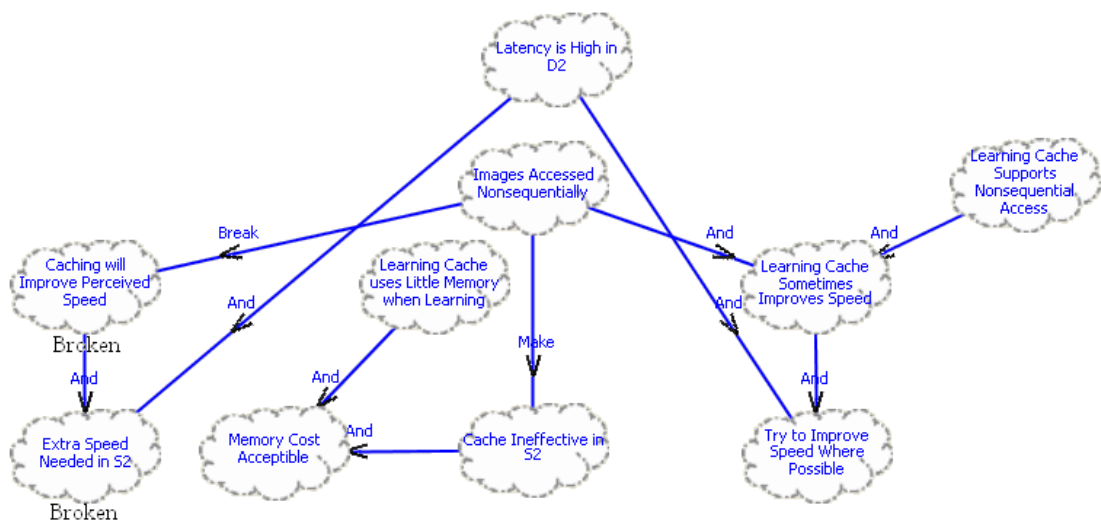


Figure 21: Complete Claim Refinement Model Supporting Learning Cache's Use

The three valid bottom-level claims depicted in Figure 21 can be added to the  $S_2$  target system's SR model to justify the selection of the learning cache to satisfy the “Load Image Data” goal, as depicted in Figure 22.

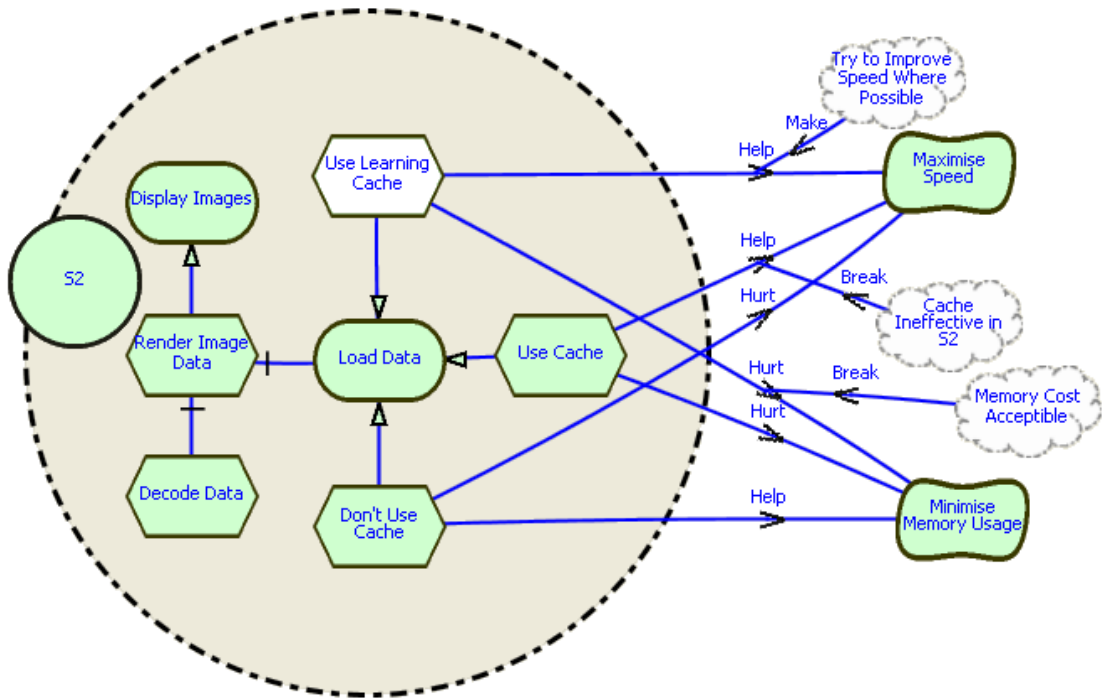


Figure 22: Adaptive Image Viewer's  $S_2$  Target System with Learning Cache Rationale

Complexity in  $i^*$  models is a recognised problem [104] [105] [106], and adding three claims to a model where a single claim would be sufficient to break the deadlock on the model and represent a decision may seem frivolous. There is thus a tension between accurately recording the rationale behind a decision, and minimising the complexity (or maximising the comprehensibility) of the models. This problem is discussed further in 6.2, but for now Figure 23 and Figure 24 depict a less precise rationale for the decision captured in a single claim on the SR model, and the corresponding Claim Refinement Model respectively.

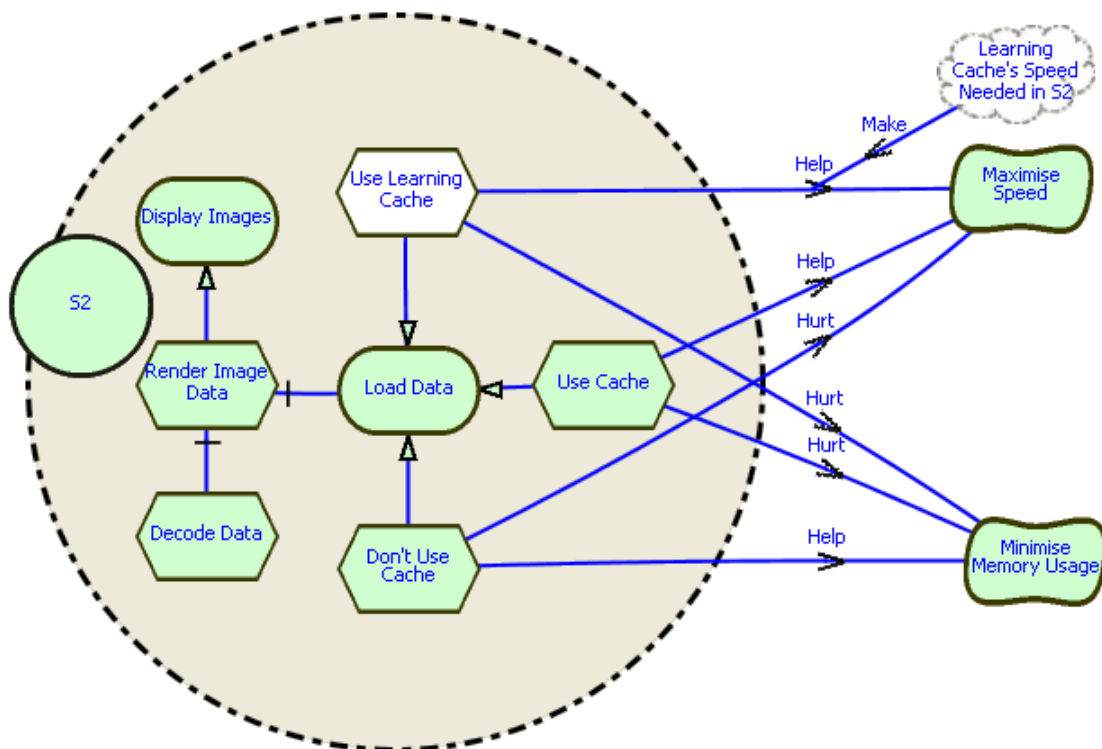


Figure 23: Simplified SR Model Showing Learning Cache's Selection in  $S_2$

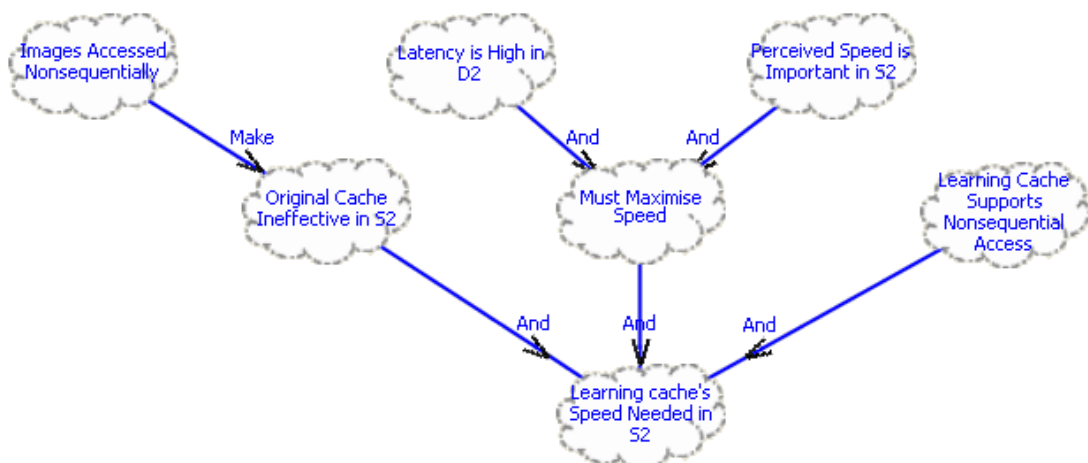


Figure 24: Simplified Claim Refinement Model Supporting Learning Cache's Use

### Consequential Change

A consequential change is one brought about as a result, either directly or indirectly, of another change. This type of change is most commonly encountered when dealing with a requirement with an overall budget such as a maximum weight or maximum power consumption. For the adaptive image viewer, no such budgeted requirement exists, so for the purposes of this example, we shall introduce the classic budgeted requirement: cost. The decision in the previous example to create a learning cache better suited to the  $D_2$  domain comes at a cost in terms of development time and, thus, cash. If the project to develop the adaptive image viewer had an overall budget, it may be that this decision takes the whole system over budget. Figure 25 shows the  $S_2$  target system's SR model with this example budgeted requirement added.

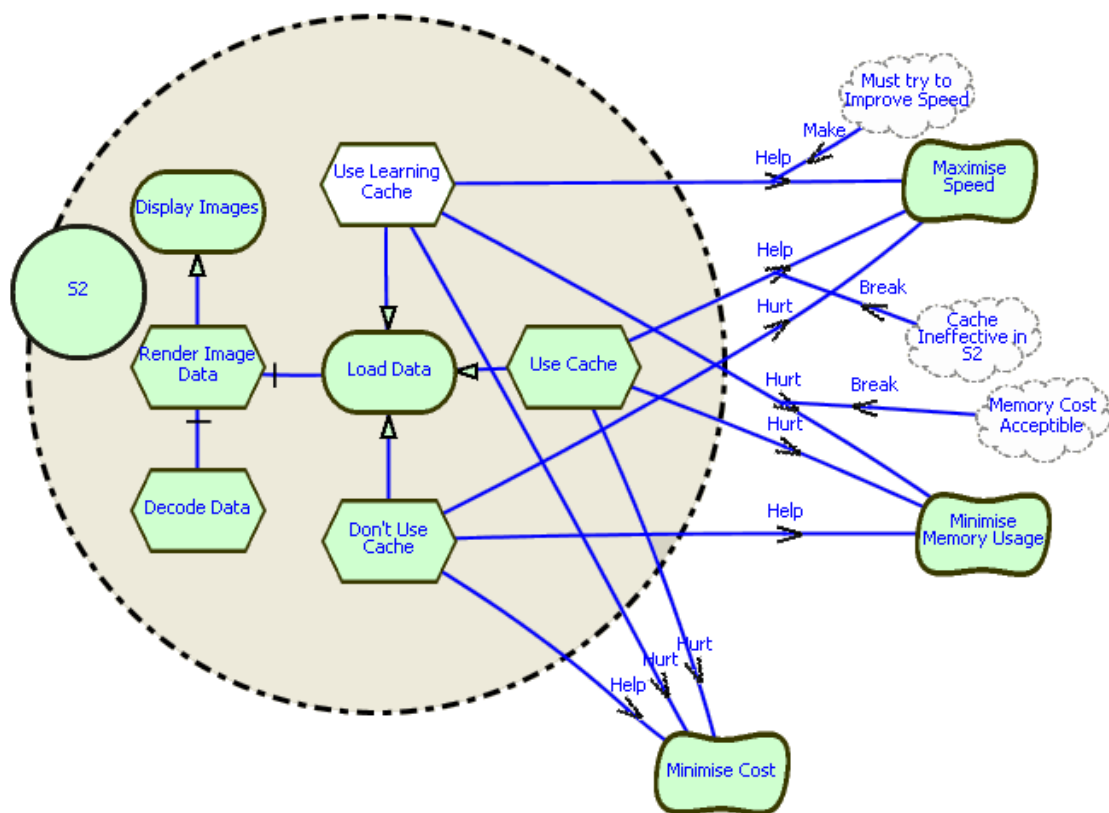


Figure 25: Adaptive Image Viewer's  $S_2$  Target System with Cost Contributions

Merely adding the “Minimise Cost” softgoal and the necessary contribution links has the effect of visually tilting the decision process in favour of not using a cache at all, given that both of the other options for the “Load Image Data” goal have two negative contributions to one positive. However, the need to maximise perceived speed is valued highly, and cost has been largely ignored in the decision. This could be represented by a “Cost Acceptable” claim, but as this is not the focus of the example, we shall instead turn to the issue of handling the fact that the system is now over budget.

Dealing with a consequential change involves tracing across target systems all of the decisions taken that affect the budget under consideration. In the adaptive image viewer, there is only a single decision taken in each target system: the decision whether or not to use a cache (or now the new learning cache). Examining each of these decisions in turn to see if a change can be made to bring the system back down to budget may yield new decision outcomes, with appropriate rationale. For the adaptive image viewer, however, the decisions are unlikely to be reversed and thus the only option available is to re-negotiate the budget.

In some situations, after tracing the decisions that affect a budget it would be possible to identify likely candidate decisions for adjustment using softgoal priorities. However, correctly setting priorities is no easy task, and to do so risks offering the illusion of quantitative precision for what are in reality imprecise qualitative expressions of priority. Furthermore, decisions often do not adhere strictly to a set of priorities, and attempting to impose some priority metric on the affected softgoals will give the appearance of contradictory and inconsistent decision outcomes. Some seemingly contradictory decisions cannot even be represented by assigning softgoal priorities, as discussed previously in Section 4.2.3.

### Stakeholder Requirements Change

A stakeholder requirements change may occur at any time during the specification, design, or implementation of the system, or indeed after deployment with changes required during maintenance. The scope and impact of this type of change is impossible to predict. A change may range from a simple re-prioritisation of previously identified requirements; spurring a need to revisit the trade-offs made involving them, or a newly introduced goal for the system to achieve. As an example, the former case is discussed below.

The core of the adaptive image viewer's need for adaptation is that the relative priorities of the “Maximise Perceived Speed” and “Minimise Memory Usage” softgoals vary in differing environmental conditions, and that no one balance can be reached that is acceptable across all the sets of environmental conditions in which the system is expected to operate. Thus, a shift in the relative priorities of these requirements makes a significant change to the desired behaviour of the system overall. To illustrate this, the priorities of the two softgoals in the  $S_1$  target system shall be adjusted.

It is discovered that, when operating in low network latency conditions ( $S_1$ ), the other applications running on the adaptive image viewer's hardware require far less memory than previously anticipated. Thus, the need to conserve memory in  $S_1$  is diminished. Unfortunately, this is insufficient to allow the use of the previously discussed learning cache in  $S_1$ : it simply uses too much memory. In  $S_2$ , the image viewer has been adjudged to have a greater need for memory than the other applications, and thus the cost in terms of memory of the learning cache remains acceptable. Returning to  $S_1$ , Figure 26 shows the previously reached specification decisions.



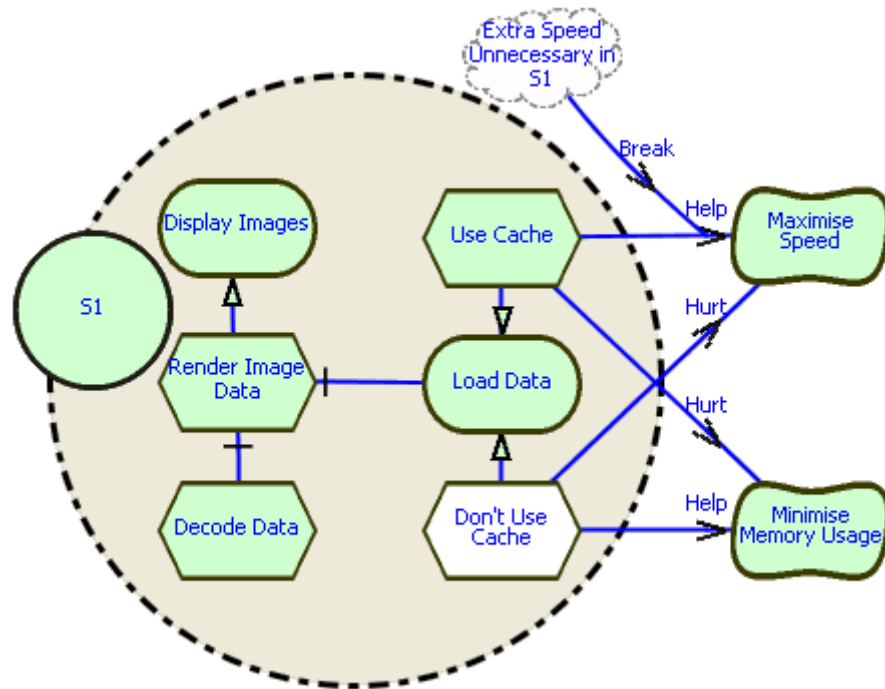


Figure 26: ReAssuRE Model of Adaptive Image Viewer's  $S_1$  Target System

The previous rationale justifying the decision not to use cache is identified as being that the speed increase is unnecessary in  $S_1$  (and thus the cost in terms of memory usage is unjustified). When the learning cache is introduced into the  $S_1$  target system model, the relative affordability of each of the cache's memory usage can be captured with claims as shown on Figure 27. The associated Claim Refinement Model, in this case, would feature claims of the available memory, and the expected consumption of each type of cache to support the decision.

Of course, this example is highly contrived, its presence shows merely that *some* stakeholder requirements changes *may* be more easily handled with the availability of decision rationale, which was eluded to by the “indeterminate” in Table 2 on page 77. The potential scope of stakeholder requirement changes varies significantly, and thus it is infeasible for a model to offer supporting traceability information to cover all scenarios.

Page 95

## 5.2 Policy Derivation

The research objectives presented in Section 1.3 included answering the following question:

To what extent is a DAS's adaptive behaviour merely a derivation of environmental analysis and configuration decisions?

If the answer to this question is “entirely”, one way to prove so is to demonstrate an ability to derive the policies controlling the DAS's adaptive behaviour from the models codifying the environmental analysis and configuration decisions, such as ReAssuRE models. Adaptation policies state the trigger condition for an adaptation, as well as the reconfigurations performed to achieve it. Thus, from a policy-derivation perspective, there are two key pieces of information to ascertain from the models:

1. the differences in decision outcomes between target systems and
2. the trigger condition that prompts the transition between target systems

The first piece of information can be obtained by comparing the Level-One SR models of the appropriate target systems. The Level-One SR models for each of the adaptive image viewer's two target systems, first presented in Figure 26 and Figure 4, on pages 94 and 54, respectively, are depicted side-by-side in Figure 28.

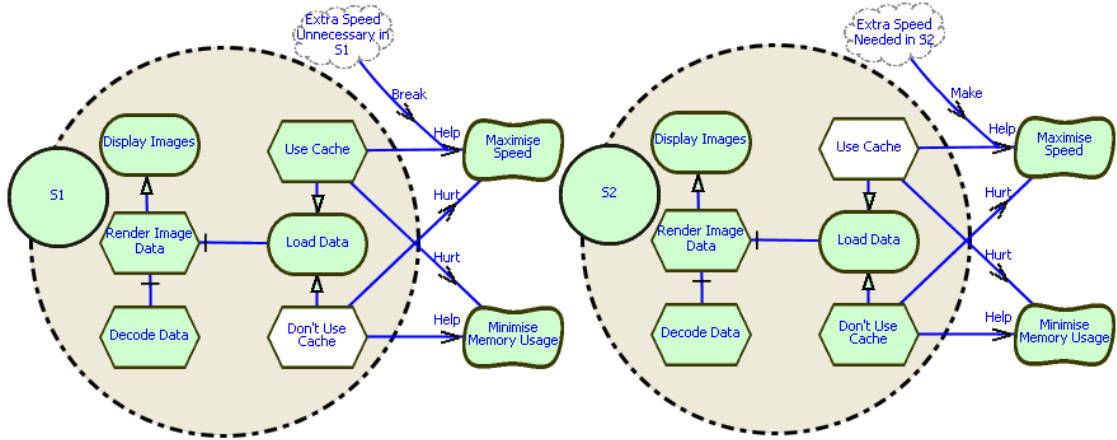


Figure 28: ReAssuRE Models of Adaptive Image Viewer's Two Target Systems

By comparing the two Level-One SR models in Figure 28, it is possible to infer the component substitutions involved in transitioning from  $S_1$  to  $S_2$  and *vice versa*. The precise component (or class) names associated with each selectable task can be placed in a lookup table when generating policies. As Figure 28 shows, the only difference between the two target systems is the means by which the “Load Data” goal is satisfied. In the  $D_1$  domain,  $S_1$  satisfies the goal with the task: “Don't use Cache”, and in  $D_2$ ,  $S_2$  opts for “Use Cache”. Thus, to transition from  $S_1$  to  $S_2$ , the adaptation policy must specify that the component associated with “Don't Use Cache” is disabled, and that associated with “Use Cache” is enabled. For the reverse transition, the inverse is true.

The second piece of information, the condition which triggers the adaptation, is ascertained by examining the Level-Two model associated with the transition. In the ReAssuRE process, a Level-Two model is produced for each valid transition, so the presence or absence of a model for a transition speaks to the need to produce policy rules to specify it.

The Level-Two model for the  $S_1$  to  $S_2$  transition was depicted in Figure 11 on page 68 (Section 4.3). As discussed in Section 4.3, ReAssuRE Level-Two models detail three different elements of a DAS's adaptive infrastructure: the monitoring mechanism, which is responsible for monitoring environmental parameters, aggregating the data and providing

it to the decision-making mechanism. The decision-making mechanism uses the monitoring data to determine when adaptation is necessary, using an adaptation trigger to signal the adaptation mechanism, which effects the adaptation. Clearly, it is the adaptation trigger which is of interest here, and it can be identified in Figure 11 as the “Fire HIGH\_LATENCY Event” task. Thus, the two pieces of information highlighted as required at the start of the section, for the  $S_1$ - $S_2$  transition, are:

1. “Load Data” being satisfied by “Use Cache” instead of “Don't Use Cache” and
2. the adaptation trigger: the firing of the “HIGH\_LATENCY” event

These two pieces of information are sufficient to write an Event-Condition-Action (ECA) rule for the reconfiguration. Figure 11 shows the ECA rule for the  $S_1$ - $S_2$  transition as a snippet of XML, as used by the GridKit adaptive middleware [39] to specify adaptive behaviour.

```
<ReconfigurationRule>
  <FrameWork>Cache</FrameWork>
  <Events>
    <Event><Type>HIGH_LATENCY</Type><Value/></Event>
  </Events>
  <Reconfiguration>
    <FileType>Java</FileType>
    <Name>Reconfigurations.Cache</Name>
  </Reconfiguration>
</ReconfigurationRule>
```

*Figure 29: Snippet of Adaptive Image Viewer's Adaptation Policy*

The snippet of adaptation policy depicted in Figure 29 specifies the so-called configuration framework [39] used by the reconfiguration, which in this case is the

“Cache” framework. A configuration framework abstracts over the different possible combinations of components in use by the DAS (or a subsystem) at a particular time, and presents a common, stable, interface to the business logic developers.

The “Events” section of the policy specifies the event(s) that will trigger the adaptation. Events may trigger adaptation merely by being fired (as in this case), or may only do so if a specific value is associated with the event. The “Reconfigurations” section of the policy specifies which reconfigurations must be performed as part of the adaptation. In this case, the only reconfiguration performed enables the file cache.

The process of policy derivation is amenable to automation, which can potentially reduce the time taken writing adaptation policies, which is particularly advantageous if a DAS's specification is liable to change over time, as argued in Section 5.1.1. Automation also reduces the scope for error during the derivation process, which increases confidence that the final adaptive behaviour of a DAS corresponds to the outcomes of decisions taken during early-phase RE.

High-level, descriptive pseudocode for the algorithm used to perform the policy derivation is included as Appendix D – Pseudocode for Policy Generation. However, the algorithm as implemented for the tool support described later in this section, makes two important assumptions about the supplied Level-Two model:

1. The Level-Two model has a “Fire Event when Environment Changes” task.
2. The Trigger task is connected to the above task, and the task label begins with the word “Fire”

Thus, the tool support presented in this subsection for policy generation is dependent on the supplied Level-Two model conforming to a similar template as that illustrated in Figure 11. Although all the Level-Two models encountered during the preparation of this thesis have been near identical, and have met the two requirements above, it may be that

some DAS models will not confirm to this template. In such cases, a new trigger-finding algorithm will need to be found to allow the policy derivation method to function.

To demonstrate automated policy derivation, this thesis presents a command-line tool capable of producing adaptation policies compatible with the GridKit adaptive infrastructure [39] from a collection of ReAssuRE models using the method discussed above. The tool operates on saved ReAssuRE models produced using the OME3 i\* modelling tool [84], and takes command-line arguments as specified in Table 3 below:

Command Line Parameter	Purpose
-l1	Comma-separated list of Level-One models
-l2	Comma-separated list of Level-Two models
-f	Configuration framework name
-t	Class name lookup table
-o	Output file name for generated policy

*Table 3: Command-Line Arguments of Policy Generation Tool*

The “l1” and “l2” parameters specify the file names of the ReAssuRE models from which the adaptation policy is to be generated. The “f” parameter specifies the name of the configuration framework used by the policy. The “t” parameter specifies a text file containing class names and associated model element names, which allows human-readable element names to be used in the ReAssuRE models and specific class names to be used in the adaptation policy. Finally, the “o” parameter specifies the file name to which the generated policy should be written.

For illustrative purposes, a screen-shot of the tool performing the policy generation is included below as Figure 30. The complete adaptation policy for the Adaptive Image Viewer, as generated by the tool is included in full as Appendix A – Image Viewer's Adaptation Policy.

```

C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>PolicyConverter
-Image.bat

C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>java -jar Polic
yConverter.jar -l1 ImageViewerModels/ImageViewer-S1.tel,ImageViewerModels/ImageV
iewer-S2.tel -l2 ImageViewerModels/ImageViewer-S1-S2-Lvl2.tel,ImageViewerModels/
ImageViewer-S2-S1-Lvl2.tel -t lookup.txt -f Cache -o ImageViewer.xml
Debug: Model Built from ImageViewerModels/ImageViewer-S1.tel
Debug: Model Built from ImageViewerModels/ImageViewer-S2.tel
Debug: Model Built from ImageViewerModels/ImageViewer-S1-S2-Lvl2.tel
Debug: Model Built from ImageViewerModels/ImageViewer-S2-S1-Lvl2.tel
Debug: Class Name Lookup Table Populated with 12 entries.
Debug: Policy Conversion Successful. Time spent: 281 milliseconds.

C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>_

```

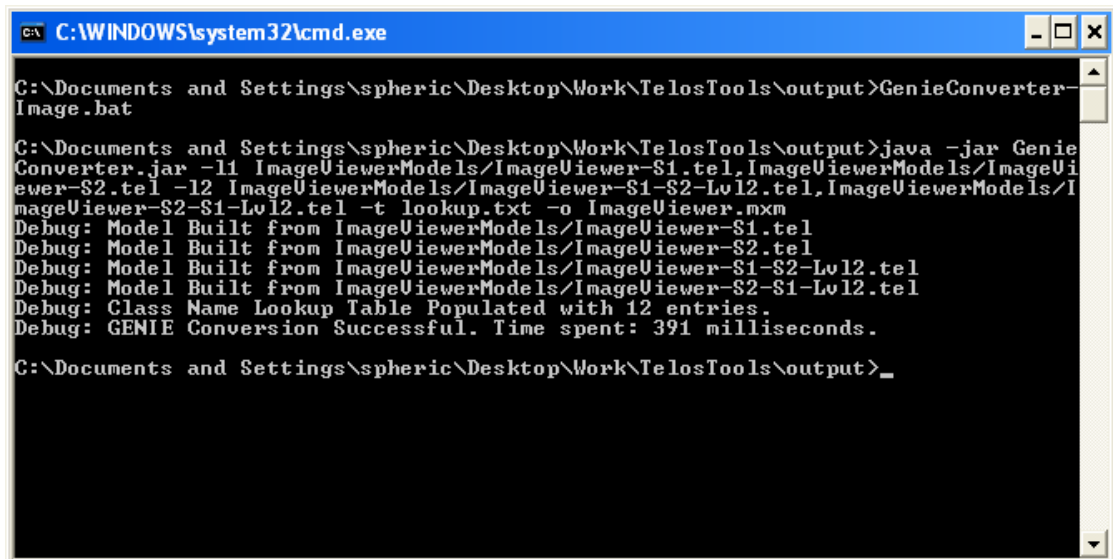
*Figure 30: Screen-Shot of Policy Generation Tool*

Although an ability to derive policies for a specific middleware implementation from ReAssuRE models is sufficient to demonstrate a clear link between the decisions codified in the models and a DAS's eventual adaptive behaviour, the utility of the tool is restricted to any DAS using that specific middleware as an adaptive infrastructure. To combat this restriction, the tool has a second mode of operation, which instead of generating policies for a specific adaptive middleware; outputs the derived adaptive behaviour to a domain specific language designed specifically for this purpose.

The Genie Domain Specific Language (DSL) [107] allows adaptive behaviour to be specified and expressed independently of adaptive infrastructure, and there is also a visual modelling tool based on the MetaEdit+ modelling meta-tool [108] available to allow adaptive behaviour expressed in the Genie DSL to be manipulated visually. Adaptive behaviour expressed in the Genie DSL can be exported from the tool in the format required by one of several adaptive infrastructures, as detailed in [107].

Figure 31 and Figure 32 depict the tool transforming adaptive behaviour from ReAssuRE models to the Genie DSL, and the resultant model exported from the Genie visualisation tool, respectively.





```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>GenieConverter-
Image.bat
C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>java -jar Genie
Converter.jar -l1 ImageViewerModels/ImageViewer-S1.tel,ImageViewerModels/ImageVi
ewer-S2.tel -l2 ImageViewerModels/ImageViewer-S1-S2-Lvl2.tel,ImageViewerModels/I
mageViewer-S2-S1-Lvl2.tel -t lookup.txt -o ImageViewer.mxm
Debug: Model Built from ImageViewerModels/ImageViewer-S1.tel
Debug: Model Built from ImageViewerModels/ImageViewer-S2.tel
Debug: Model Built from ImageViewerModels/ImageViewer-S1-S2-Lvl2.tel
Debug: Model Built from ImageViewerModels/ImageViewer-S2-S1-Lvl2.tel
Debug: Class Name Lookup Table Populated with 12 entries.
Debug: GENIE Conversion Successful. Time spent: 391 milliseconds.
C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>
```

Figure 31: Screen-Shot of the Genie DSL Derivation Tool

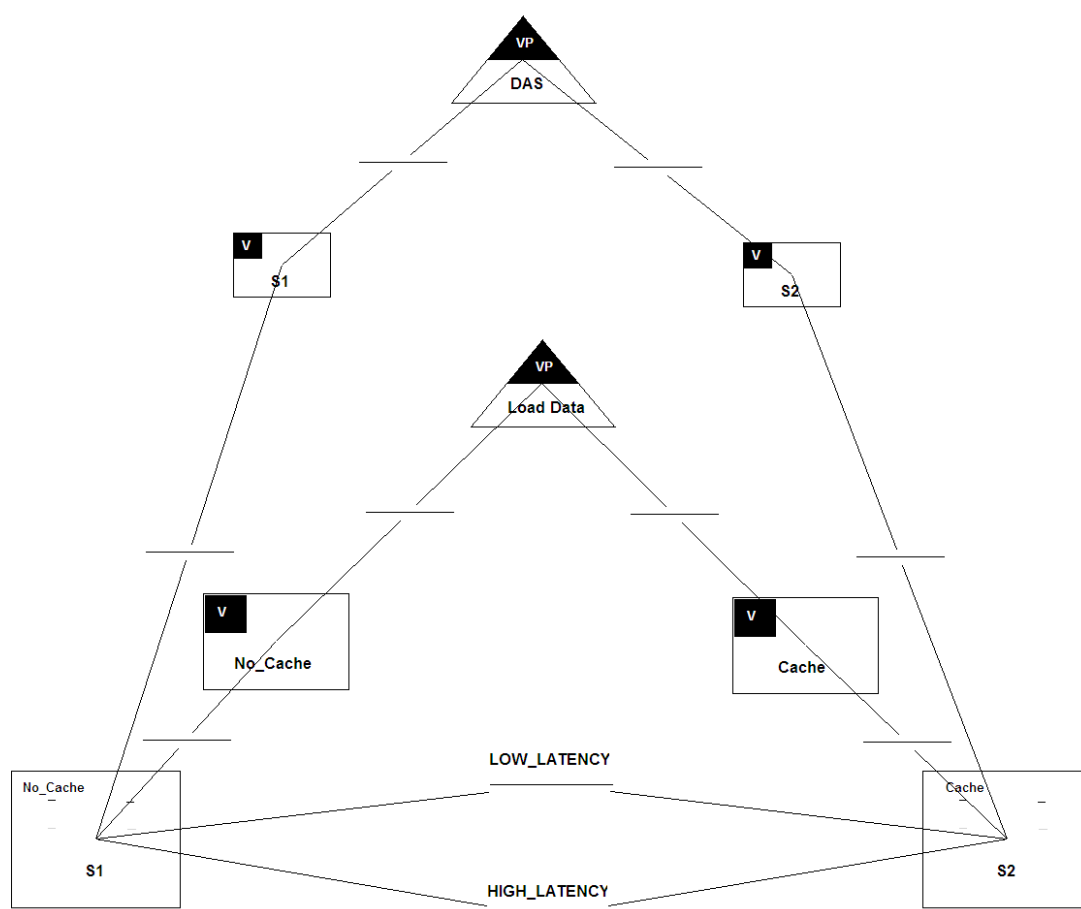


Figure 32: Genie model of Adaptive Image Viewer's Adaptive Behaviour

The Genie model depicted in Figure 32 takes the form of a feature model [109], as frequently encountered in software product line circles. Feature models illustrate the combinations of features that different related software products within a family (a product line) provide. By way of an analogy, a DAS can be considered as a product line, with each individual target system considered a product [110]. Each product (target system) select different combinations of features (decision outcomes, or individual components) from those available.

The triangles in Figure 32 marked with “VP” represent a variation point, where a choice is made to specify (or omit) a feature in an individual product within the product

line. The rectangles marked with a “V” are variants, representing a feature within the product line. The squares at the bottom of Figure 32 represent individual products, and represent DAS target systems in GENIE models. The DAS in Figure 32 is itself a variation point, capable of using the  $S_1$  or  $S_2$  feature (target system). The other variation point, “Load Data”, corresponds to the identically named task in the two Level-One SR models in Figure 28, and can be specified using either of the linked variants: “No Cache” or “Cache”.

To conclude, this section has demonstrated it possible to derive a DAS's adaptation policies from ReAssuRE Level-One and Two models directly and automatically. The environmental analysis and configuration decisions codified in the ReAssuRE models are transformed into a DAS's adaptive behaviour, which directly answers one of the research questions posed in Section 1.3:

To what extent is a DAS's adaptive behaviour merely a derivation of environmental analysis and configuration decisions?

As such, it is reasonable to answer the research question: “entirely”. The automation of the method, as demonstrated by the tool support introduced, is also sufficient to answer a second research question posed in Section 1.3:

Given that both the information from the environmental analysis and the configuration decisions are subject to change, how can the workload of deriving a DAS's adaptive behaviour be reduced?

The workload involved in deriving a DAS's adaptive behaviour manually is significant, and the process potentially error-prone. If, as argued in Section 5.1.1, a DAS's specification is liable to change, the cost associated with re-deriving adaptive behaviour potentially multiple times in response to changes could become prohibitive. The automation presented in this section significantly reduces the time taken to re-derive adaptive behaviour, and eliminates the possibility of human error from the derivation process. This policy derivation work has also been reported in preparation for this thesis [111].

## 5.3 Requirements Validation

Specifying a system capable of adapting its behaviour to suit a volatile environment is a difficult task, and any eventual specification should be subject to significant validation effort to ensure that the behaviour specified for given sets of environmental conditions is appropriate. Claims in ReAssuRE models can be used to highlight areas of uncertainty in the reasoning behind a DAS's specification, and by developing validation scenarios seeking to explore the likelihood and consequences of deficiencies. This section introduces a method by which key validation scenarios can be identified using ReAssuRE models, supporting DAS requirements validation, as previously reported by Welsh & Sawyer [88].

Claims in ReAssuRE models can vary in the degree of certainty held in them by the analyst. Some claims are axiomatic, others little more than assumptions that were needed to reach a decision. If a claim is wrong, the performance of the DAS may be unsatisfactory, the DAS may exhibit harmful emergent behaviour, or the system may fail completely. As such, claims should be validated where possible to ensure their correctness. Unfortunately, the complex and uncertain nature of the environments DASs are developed for, along with the likelihood of analysts working with incomplete knowledge, means that some claims will prove difficult if not impossible to validate with complete assurance during the RE process. In these cases, it is beneficial to examine the behaviour of the DAS if a claim is proven false.

ReAssuRE models allow the analyst to examine the potential impact of a claim's falsity, by tracing the impact throughout models. In cases where a bottom-level claim (that is, a claim underpinning a decision on a Level-One SR model) is affected by a claim's falsity, some action may be necessary to better understand a DAS's behaviour if such a scenario manifested itself. This thesis uses the term *validation scenario* for such activity, but does not prescribe the form of a validation scenario. Most obviously, validation scenarios could take the form of test cases executed to verify the DAS's behaviour in the circumstances covered by the validation scenario complies with set criteria. However, in some circumstances simulation, modelling or static reasoning may be more appropriate.

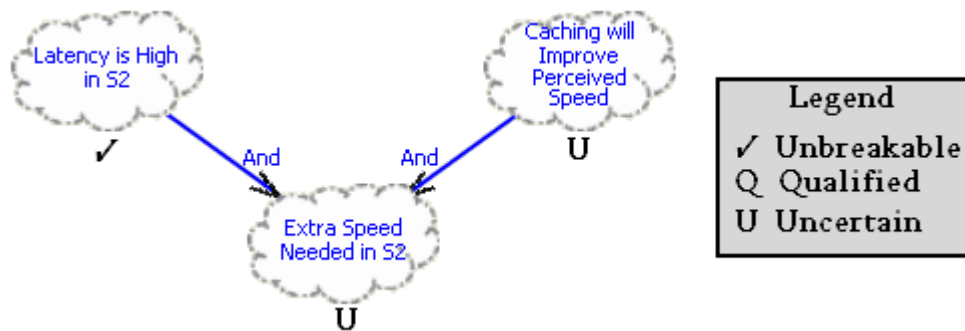
In keeping with the qualitative nature of the reasoning supported by  $i^*$ , the ReAssuRE process supports reasoning with the degree of uncertainty about a claim in a simple qualitative manner. A claim that is axiomatic is afforded the label **unbreakable**, whilst a claim in which a low degree of confidence is held is termed **uncertain**. These two labels are two extremes of the confidence scale, a claim that lies between the two extremities is termed **qualified**. In borderline cases in which it is difficult to classify claims as uncertain or qualified, the consequences of claim falsification may be taken into account, with borderline claims classified as uncertain if the consequences of the claim not holding appear serious.

Returning to the Claim Refinement Model for the  $S_2$  target system, first seen in Figure 6 on page 57, allows the claims to be classified and assigned confidence labels. Assigning confidence labels to all but the bottom-level claim on Figure 6 and propagating them throughout the model allows us to establish the degree of confidence held in the bottom-level claim. The “Latency is High in  $S_2$ ” claim is axiomatic and thus should be labelled unbreakable: high latency is the defining feature of  $S_2$ . The “caching will improve perceived speed” claim, on the other hand, is uncertain: as we saw in Section 5.1.2, images being accessed non-sequentially could easily render the claim invalid.

To propagate the uncertainty labels throughout the model, rules similar to those used in Section 5.1.2 on page 82 to propagate “broken” labels throughout a claim refinement model are used, these are:

- If a claim is derived from a single claim, then the derived claim inherits the label of the upper-level claim.
- If a claim is derived from two or more AND-ed claims, then it inherits the *least certain* label of its supporting claims.
- If a claim is derived from two or more OR-ed claims, then it inherits the *most certain* label of its supporting claims.

A bottom-level claim that, once label propagation has taken place, is classed as unbreakable has little uncertainty underpinning it. A bottom-level claim classed as uncertain or qualified has a higher degree of uncertainty, and should be the target of validation work, especially so in the case of an uncertain bottom-level claim. Applying the rules above to propagate uncertainty labels throughout Figure 6 yields Figure 33.



*Figure 33: Uncertainty Propagation in  $S_2$  Claim Refinement Model*

The bottom-level claim in Figure 33 has been classified as uncertain. As such, there is a need to explore how the DAS would behave in circumstances where the claim doesn't hold. Developing a validation scenario where the behaviour of the DAS can be simulated, examined or tested becomes a matter of identifying a set of environmental conditions in which a given combination of bottom-level claims hold (or do not hold). For the adaptive image viewer, such a validation scenario would likely reveal that in such circumstances the adaptive image viewer uses memory to facilitate caching whilst deriving no benefit in terms of perceived speed from doing so.

If the outcome of a validation scenario is acceptable, then there is likely no need to refine the DAS's requirements specification for the target system under study, and some uncertainty has been mitigated. If the outcome is more problematic, this signifies a need to revisit the specification of the target system under study, and the assumptions the specification is based upon.

For target systems with greater numbers of bottom-level claims, it becomes necessary to consider a validation scenarios for each possible combination of falsified and held bottom-level claims, which can amount to a significant validation burden as the number of required validation scenarios ( $T$ ) increases with the number of bottom-level claims ( $n$ ) with the formula  $T=2^n-1$ . One is subtracted because the combination in which all claims hold is the envisaged domain.

When dealing with a large number of bottom-level claims, there is clearly a benefit in targeting validation resources at uncertain claims (and combinations including uncertain claims), as scenarios involving them are more likely to occur. The explosive increase in required validation scenarios as the number of bottom-level claims rises, along with the fact that only a sub-set of a model's bottom-level claims will be considered uncertain means that this targeting can yield a significant reduction in the number of validation scenarios required, as demonstrated in Section 7.5.

The act of deriving validation scenarios from combinations of held and falsified bottom-level claims rather than from combinations of underlying claims greatly reduces the number of validation scenarios that need to be constructed by grouping together all combinations of broken underlying claims that have a specific effect, i.e. breaking a given bottom-level claim,. Furthermore. The procedure allows scenarios rendered irrelevant by axiomatic claims to be removed from consideration.

To conclude, this section has introduced a method of classifying claims based on the degree of certainty held in them by the analyst. It has also demonstrated how uncertainty labels propagate throughout claim refinement models, and how validation scenarios may be derived from a particular combination of bottom-level claims holding or otherwise to aid in DAS requirements validation. The work presented in this section has been published in full [88].

## 5.4 Chapter Conclusion

Although a DAS possesses a limited ability to adjust its behaviour at run time in response to changes in the operating environment, the complexity of the domains for which DASs prove most useful, coupled with the likely imperfect understanding of both the environment and candidate components during early-phase RE means that the specification for a DAS is liable to change.

ReAssuRE models support change by recording additional tracing information in the form of claims, which capture the rationale behind decisions and the assumptions on which the rationale is based. This information eases the process of change in common scenarios, improving the efficiency of the change process.

A DAS's adaptive behaviour may be derived from its ReAssuRE models, and the derivation method can be automated. A tool that transforms ReAssuRE models into either adaptation policies directly, or to a DSL designed to represent adaptive behaviour has been introduced, which can save a significant amount of time during the derivation process, as well as reduce error.

Thus, the bulk of the time spent in the change process is spent updating ReAssuRE models rather than re-specifying adaptive behaviour. The improved traceability support of ReAssuRE models minimises the time taken in performing these updates.

The chapter has also answered three of this thesis's research questions, which were presented in Section 1.3:

Can the information from the environmental analysis and configuration decisions be codified in models, and is it useful to do so?

Information from the environmental analysis and configuration decisions taken during early-phase RE can be codified in models, with ReAssuRE models containing more of this information than previously possible. The utility of doing so is demonstrated by the enhanced traceability support discussed in Section 5.1.2.



To what extent is a DAS's adaptive behaviour merely a derivation of environmental analysis and configuration decisions?

A DAS's adaptive behaviour is entirely a derivation of the environmental analysis and configuration decisions taken during early-phase RE, as demonstrated by the ability to both perform and automate this derivation from the ReAssuRE models, which codify the results of environmental analysis and configuration decisions.

Given that both the information from the environmental analysis and the configuration decisions are subject to change, how can the workload of deriving a DAS's adaptive behaviour be reduced?

The workload associated with deriving a DAS's adaptive behaviour is significantly reduced by automating the derivation process. The workload associated with re-deriving adaptive behaviour in response to a specification change is reduced by the enhanced traceability support of ReAssuRE models which helps trace the impact of change in a DAS's specification, as well as by the automated derivation tool.

The need to validate requirements for a DAS is particularly acute given the relative uncertainty of the environments in which they operate. The chapter has demonstrated how claims in ReAssuRE models act as markers for uncertainty about the environment or expected behaviour, and demonstrated how reasoning with a simple scheme of uncertainty labels allows assumptions to be grouped by the effect of their falsification, and for the effects themselves to be explored using validation scenarios.

The next chapter builds upon the work discussed in this, focussing on how ReAssuRE models can be used at run time, by a DAS itself.

## 6 ReAssuRE at Run Time

---

The previous chapter discussed uses of ReAssuRE models at design time, demonstrating their improved traceability support and an ability to derive adaptation policies from the models directly. This chapter focusses on using the same models at run time to guide the DAS's adaptation.

For some time, systems have been able to monitor their performance, and the degree to which key requirements are satisfied [93] [95]. This monitoring data would typically be used by developers for tuning and maintenance purposes. The emergence of adaptive infrastructures allowing systems to adjust their behaviour at run time in response to monitoring data has bred research interest in systems that use requirements models to help interpret monitoring data, with the adaptive infrastructure providing a means to effect behavioural change to act upon it. This emerging practice is referred to as *models@run.time* [112]. This chapter demonstrates the use of ReAssuRE models at run time to allow a DAS to adjust the configuration of a target system in circumstances where monitoring data indicates the current configuration is sub-optimal.

The tool support for adaptive policy generation discussed in Section 5.2 operates by loading, parsing and reasoning with ReAssuRE models programmatically. The same techniques can be leveraged by a DAS to reason with ReAssuRE models at run time.

The ability to reason-with ReAssuRE models at run time, coupled with an ability to modify the models in response to monitored environmental conditions offers the possibility of a DAS performing some of the specification changes demonstrated in 5.1.2 autonomously. The chapter introduces a class of DAS called a Model-driven Dynamically Adaptive System, abbreviated to m-DAS. This class of DAS uses design-time models to guide run-time adaptation. Although the term m-DAS covers any DAS that uses models to guide adaptation, a m-DAS using ReAssuRE models and requirements monitoring techniques [30] serves as an example of the class of system, allowing this thesis to explore the implications of constructing systems with a greater degree of autonomy than before.

The chapter discusses the run-time representation of ReAssuRE models, the environmental monitoring that allows the models to be modified in memory, and demonstrates how a m-DAS may adjust its adaptive behaviour in response to these modifications. The chapter also examines the potential for damaging emergent behaviour, and demonstrates how a variant of the validation scenario derivation technique discussed in Section 5.3 can offer testing scenarios aiming to uncover potential emergent behaviour.

## 6.1 Run-Time Representation of ReAssuRE models

When implementing tool support for the policy derivation method discussed in Section 5.2, it was necessary to develop code to load, parse and reason with ReAssuRE models. As well as being necessary for tool support, this also offered the possibility of allowing a DAS to reason with ReAssuRE models autonomously. This, combined with monitoring techniques discussed in Section 6.2 could allow a m-DAS to derive a new, unspecified configuration when operating outside its foreseen operational envelope.

Three types of ReAssuRE models can be parsed and loaded by a m-DAS at run time: Level-One Strategic Rationale and Claim Refinement Models, along with Level-Two models. Level-Two models are used solely when deriving new adaptation policies, whereas Level-One SR and Claim Refinement Models are reasoned with to enable new configurations to be devised and adopted.

An  $i^*$  model can be thought of as an acyclic digraph, whose nodes are one of several defined  $i^*$  elements, and whose edges are one of several defined  $i^*$  relationships. Meaning is inferred by examining the types of elements connected together, and the type of relationship connecting them. The OME3  $i^*$  modelling tool [84] produces  $i^*$  models saved in the “.tel” (short for telos) file format, which are essentially textual representations of each of the elements and relationships in a model, along with a section on view-state, which controls layout. To reason with models, the view-state information can safely be discarded, and a logical representation of the model is constructed by instantiating objects

representing each encountered  $i^*$  element, linking them together where appropriate with specified relationships. As such, an in-memory representation of an  $i^*$  model is little more than a standard digraph, with most of the complexity lying in the ability to reason with the model.

Both policy derivation, as discussed in Section 5.2, and run-time reasoning focus on identifying goals in Level-One ReAssuRE models that may be satisfied by two or more tasks. These tasks are connected to the goal via a means-end link running from each task to the goal. Internally, these goals with alternate satisfaction means are referred to as “variation points”, and may be identified by traversing the digraph in memory to identify the pattern described.

Level-Two models are analysed to identify the trigger condition that warrants a transition between target systems. This is achieved by locating the decision-making mechanism's task: “Fire event when environment changes”, present on all ReAssuRE Level Two models. This task is decomposed into at least two sub-tasks, with one task typically concerning the identification of an environmental change. The sub-task of interest, is typically named “Fire X Event”, in which “X” is the name of the trigger event of interest. Identifying the correct sub-task is a matter of finding the task with a name in this format.

The most complex reasoning performed by a m-DAS at run time with ReAssuRE models occurs in Claim Refinement Models. The label propagation method described in Section 5.3 has been implemented for the “broken” label, which signifies that a claim no longer holds. The use and propagation of this label is discussed in the next section.

## 6.2 Assumption Monitoring and “broken” Label Propagation

As discussed in Section 1.2, the complexity of domains for which DASs are conceived coupled with uncertainty in both the nature of the domain and the behaviour of DAS components or a DAS as a whole means that a DAS's specification is liable to change. One

class of change discussed in Section 5.1.1 was signalled by a broken assumption. This thesis introduces the concept of *assumption monitoring*, which is analogous to the more established activity of requirements monitoring [30]. In requirements monitoring, the performance of the system with respect to satisfying key requirements is recorded and monitored to allow stakeholders to better understand the system's performance and to feed back into system refinements made during maintenance.

Except for some work focussing on automatically deriving monitors for quality features (typically bound together in QoS concerns [95], the means of monitoring requirements either directly or via some surrogate property are typically devised manually, on a case-by-case basis. This thesis proposes that a similar approach is adopted for monitoring the assumptions on which DAS specifications are based, by devising monitors where feasible for assumptions recorded in ReAssuRE Claim Refinement Models. It is not possible to monitor all assumptions directly. However, assumptions for which monitors can be devised, and where monitoring data suggests the assumption no longer holds, are broken, that is, have a “broken” label applied to them, and this status may be propagated to child claims.

The ability of a m-DAS to reason with models of its own behaviour offers the possibility of a m-DAS altering its configuration in response to changes in the models, including changes as a result of an assumption no longer holding. In this respect, a m-DAS devised to reason with ReAssuRE models, and which performs assumption monitoring, has the potential to adjust its behaviour not only to *foreseen* changes in the environment, but to circumstances outside those for which the m-DAS has had behaviour explicitly specified.

The basis of this ability to adjust behaviour in response to assumptions found to no longer hold is in the modification of ReAssuRE Level-One SR models in response to bottom-level claims being broken on the associated Claim Refinement Model. In essence, much of the work performed on uncovering a broken assumption in Section 5.1.2 is automated. The final step discussed in Section 5.1.2: re-taking decisions that were reliant (either directly or indirectly) on the broken assumption is particularly challenging.

Although it is possible via model analysis to identify assumptions reliant on a broken assumption, inferring the decision outcome that would have been reached had the faulty assumption not been made at design time is impossible in many cases. A particular problem is that contributions on Level-One Strategic Rationale models may become deadlocked.

To prevent deadlock, when devising a monitor for a particular assumption, the analyst can specify one of three actions to be performed on the Level-One Strategic Rationale model in the event of a bottom-level claim being broken either directly or by propagation. More drastic action reduces the chance of deadlock but increases the chance of a needless or unjustified change in configuration.

If a bottom-level claim on a Claim Refinement Model is broken, either demonstrably by analysing monitoring data, or by label propagation after a supporting claim is broken, then one of three actions can be performed on the corresponding Level-One SR model. These are, in increasing order of strength with “3” as the strongest:

1. Remove the claim from the Level One model.
2. Invert the claim's contribution to the link to which it is attached.
3. Remove from consideration the task whose selection is supported by the claim.

The risk of a stronger model modification causing the m-DAS to make an unjustified configuration change means that weaker model modifications, which appear earlier in the above list, are preferred whenever possible.

To reduce the chance of deadlock, a Level-One Strategic Rationale model and its associated claim refinement model can be constructed using many claims, so that should a single claim be broken and removed, the affected decisions rely on other claims. Unfortunately, this type of construction requires several claims per decision on the Level-

One Strategic Rationale model, each needing supporting on the Claim Refinement Model. This increases the complexity of the ReAssuRE models to such a degree as to be infeasible in some cases, signifying the need for stronger modifications.

Taking stronger action on the Strategic Rationale model when invalidating a bottom-level claim on its associated Claim Refinement Model can yield the same behaviour as a more complex lattice of claims, whilst using significantly fewer claims. However, more drastic modifications to the ReAssuRE models, especially when multiple such modifications are made to a model, can potentially lead to a configuration being adopted that is radically different from that envisaged by the m-DAS's creators. Such radically altered configurations carry a significant risk of unpredictable and possibly damaging emergent behaviour.

Clearly, the model modification used when invalidating a bottom-level claim in response to a given monitor's data needs to be chosen carefully. Each type of model modification is discussed, in turn and with an example, in the following paragraphs.

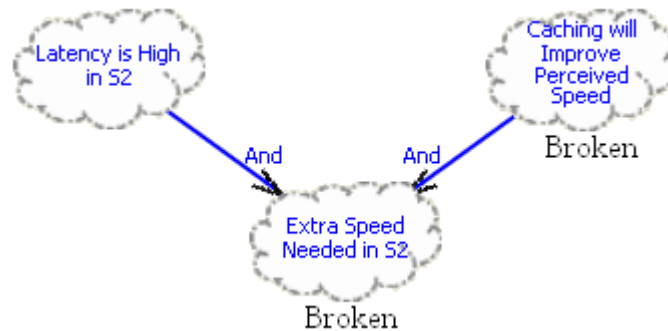
As mentioned previously, removing a broken bottom-level claim from a Level-One model is preferable. Using several claims it is possible to offer more control of the configuration the m-DAS falls back to should a claim fail. Using the “break” link in a Claim Refinement Model also allows broken bottom-level claims to be placed on a level-one Strategic Rationale model. An initially broken claim has no effect *unless* it is enabled by monitoring or propagation, and can be used to control a m-DAS's configuration should an assumption thought false at design time be found to hold at run time.

Returning to the adaptive image viewer example, review the Level-One Strategic Rationale and Claim Refinement Models for the  $S_2$  target system. These were depicted in Figure 4 on page 54, and Figure 6 on page 57, respectively.

Examining Figure 6, both supporting claims are amenable to monitoring, with monitors potentially trivially devised for both. “Latency is High in  $S_2$ ” could be monitored by measuring the latency encountered when loading files. “Caching will Improve Perceived Speed” could be monitored by measuring the time taken to display images, and

comparing it to the underlying latency encountered (as monitored for the previous assumption). However, given that high latency is a defining characteristic of the  $D_2$  domain, if latency should fall then the environment would be considered to be in the  $D_1$  domain, with the DAS adopting the  $S_1$  target system. As such, monitoring the “Latency is High in  $S_2$ ” assumption is needless.

If the “Caching will Improve Perceived Speed” monitor were to yield data suggesting the assumption doesn't hold (in Section 5.1.2, this was because images are being accessed non-sequentially) then, just as in the manual example in Section 5.1.2, a “broken” label is applied to the claim. Following the rules of propagation set out on page 106 in Section 5.3, the “Extra Speed Needed in  $S_2$ ” claim's holding depends upon both supporting claims also holding. Thus, the “broken” label propagates to the bottom-level: “Extra Speed Needed in  $S_2$ ” claim, invalidating it also. The modified claim refinement model is depicted in Figure 34 below.



*Figure 34: Modified  $S_2$  Claim Refinement Model*

Because the Level-One SR model depicted in Figure 4 has no other claims, removing the now invalidated bottom-level claim from the model will lead to the decision governing the use of the cache becoming deadlocked. It is possible to prevent this deadlock without using any stronger model modification by adding an *alternate* claim to both the Strategic Rationale and the Claim Refinement Model. On the claim refinement model, a “break” contribution can be used to force the alternate claim to start broken, and



only exert influence on the decision governing cache use if reinstated. The Level-One Strategic Rationale and Claim Refinement Models showing this construction are depicted in Figure 35 and Figure 36.

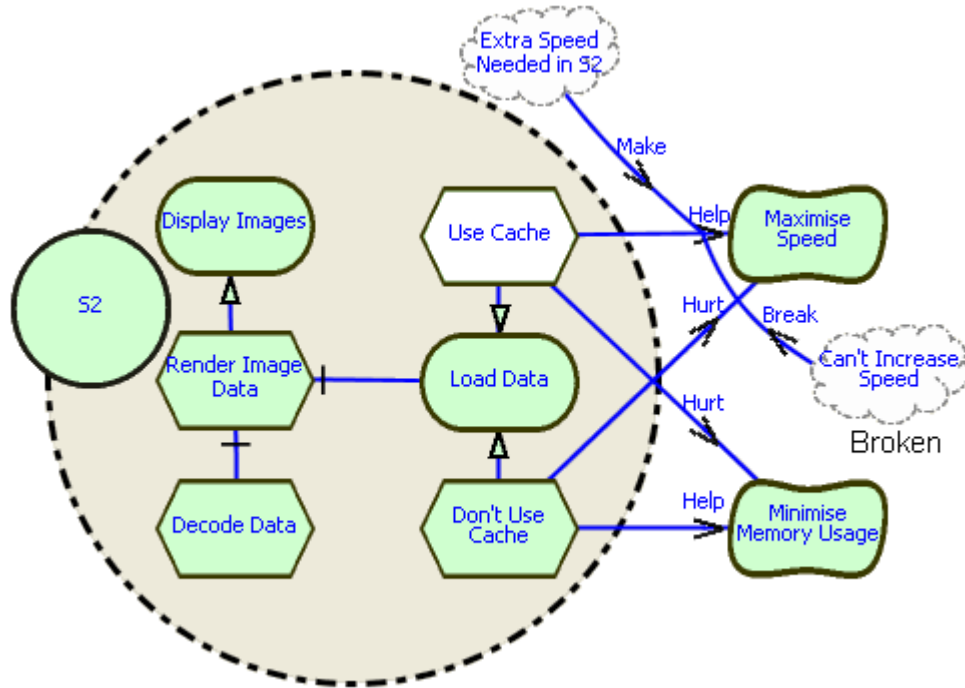


Figure 35: Level 1 SR Model of Image Viewer  $S_2$  with Alternate Claim

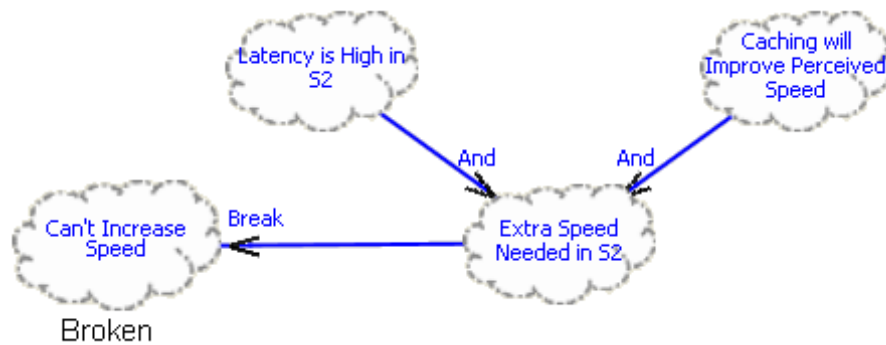


Figure 36: Claim Refinement Model for Image Viewer  $S_1$  With Alternate Claim

The “Can't Increase Speed” alternate claim in Figure 36 is broken from the outset, and has no effect on the decision whether or not to use a cache: its presence is ignored given its “Broken” label. However, should the “Extra Speed Needed in  $S_2$ ” claim be broken, the “break” link between it and the “Can't Increase Speed” claim has the effect of inverting the “Broken” label during propagation. This means that the “Can't Increase Speed” claim is reinstated on the Strategic Rationale model and the “Broken” label is removed. Once reinstated, the “Can't Increase Speed” claim breaks the “Use Cache” task's positive contribution to the “Maximise Speed” softgoal, arguing against the use of the cache. The now broken “Extra Speed Needed in  $S_2$ ” claim is removed from the SR model.

The configuration derived from the models depicted in Figure 35 and Figure 36 with the “Extra Speed Needed in  $S_2$ ” claim broken, and the alternate “Can't Increase Speed” claim reinstated, would not use the cache in the  $S_2$  target system. Given that the adaptive image viewer has only a single possible variation point, and only two possible variations. Adding alternate claims to the models could be considered overly complex. Furthermore, the decision as to which eventualities should be catered for on a model is notoriously difficult, and is analogous to the decision of which obstacles to model when performing KAOS obstacle analysis. Thus, it may be preferable to use a stronger model modification technique to avoid the scoping difficulty. Using a stronger model modification technique will yield the same adaptive behaviour in this instance, without the need for additional model complexity. Inverting the original bottom-level claim's contribution to the Level-One SR model when broken yields the model in Figure 37.

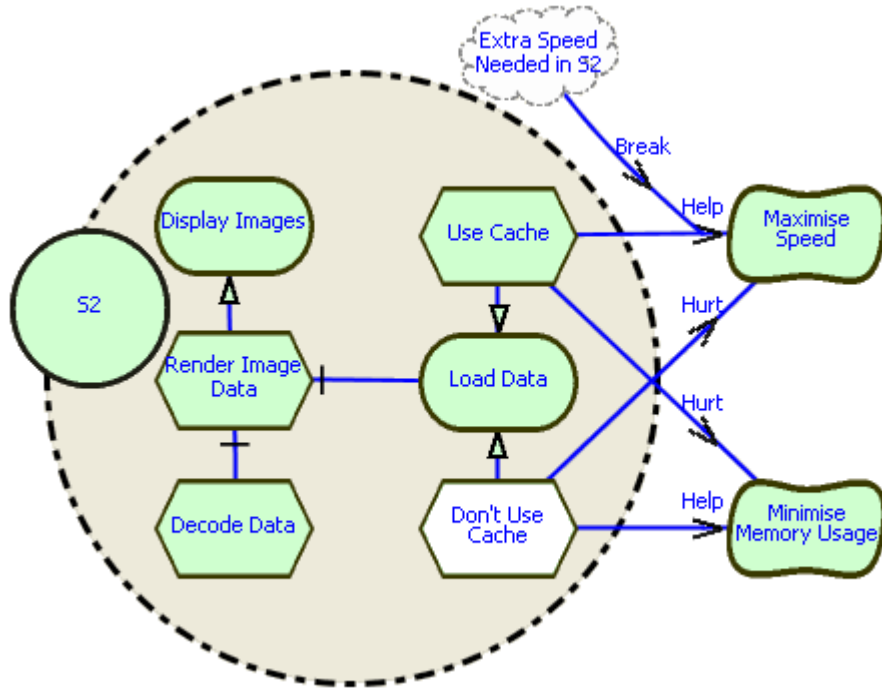


Figure 37: Level 1 SR Model of Image Viewer  $S_2$  after Claim Inversion

In Figure 37, instead of the claim “Make”ing the positive contribution the “Use Cache” task makes to the “Maximise Speed” softgoal, it “Break”s it. Now, instead of the claim supporting the use of the cache, it supports the selection of the “Don't Use Cache” task. Thus, the configuration of the adaptive image viewer's  $S_2$  target system would again, omit the cache. Note that there is no “broken” label on the claim in Figure 37, - using this modification means that there is no need to discount the claim when evaluating the model.

The strongest model modification that may be performed in response to a broken assumption is to remove the task that the claim supports the selection of from consideration when re-making the decision. This modification is suitable if and only if a claim's holding is sufficient to singularly justify the task's selection, and its absence is sufficient to render the selection of the task non-viable. Performing this modification in response to the “Extra Speed Needed in  $S_2$ ” claim no longer holding yields Figure 38.

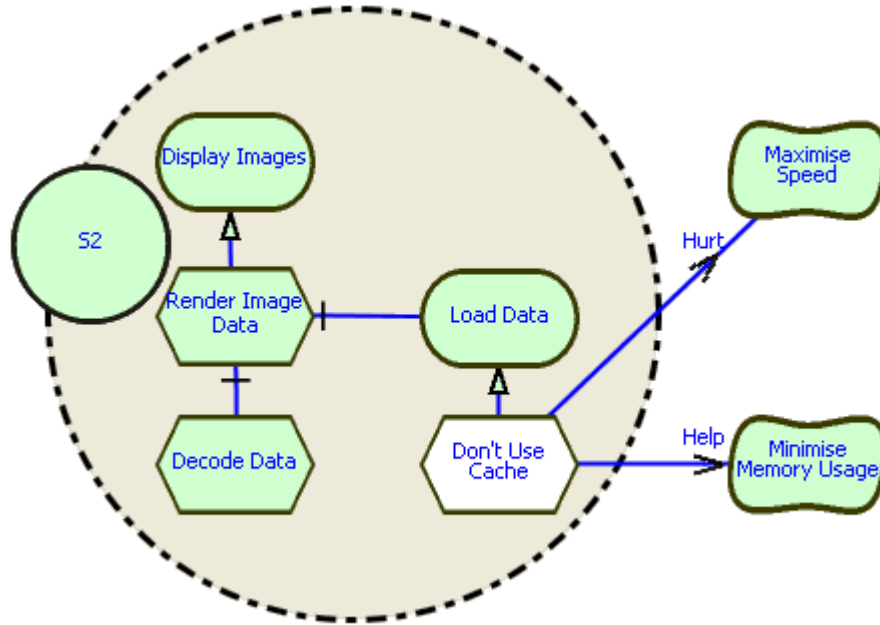
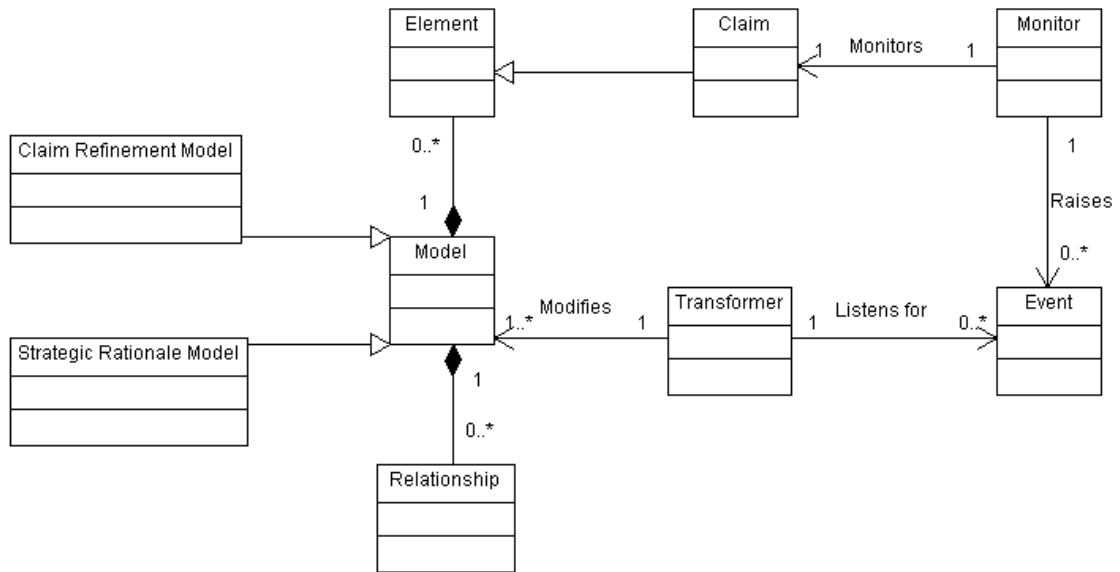


Figure 38: Level 1 Model of Image Viewer  $S_2$  after Removal Modification

The same configuration is reached following this modification. Because there are only two viable decision alternatives, the only remaining alternative is to select “Don't Use Cache”. For decisions with three or more potential tasks for selection, specifying that a task should be removed from consideration should an assumption not hold is a valuable tool in reducing the likelihood of deadlock if unexpected conditions are encountered, without complicating the ReAssuRE models.

### 6.3 ReAssuRE m-DAS architecture

As discussed in the chapter introduction, the use of the term m-DAS does not dictate that a system use ReAssuRE models to guide its adaptation. An m-DAS could also go about modifying models using means other than assumption monitoring. However, the use of ReAssuRE models and assumption monitoring to create a m-DAS implies a specific architecture, a UML model for which is depicted below.



*Figure 39: m-DAS Architecture Model*

In Figure 39, assumptions are codified as claims in models, and have monitors devised to validate them. In the event of a monitor determining that an assumption no longer holds, it raises an event. A model transformer acts on receiving the event, modifying the model. Once modified (and external to Figure 39), the policy generation method described in Section 5.2 can be used to re-derive adaptation policies for the m-DAS, with new adaptation policies dictating new target system specifications. Each of the entities depicted in Figure 39 is discussed in more detail in the following paragraphs.

The “model” entity in Figure 39 covers both Level-One Strategic Rationale and Claim Refinement ReAssuRE models, discussed in Chapter 5. Assumptions are represented by claims in the models. “Monitor”s are devised to validate claims at run time. Should a claim be invalidated by monitoring data, an “event” is raised. The “event” entity represents a simple programmatic event, that may be fired by a monitor and listened for by interested parties. In the architecture described by Figure 39, the only interested party is the model transformer. It is this “transformer” entity that is key to a ReAssuRE-based m-DAS.

The “transformer” entity's role is to make adjustments to models to reflect an assumption that no longer holds, as indicated by the monitor's event for which it listens. When acting upon a monitor event indicating an assumption no longer holds, the transformer invalidates the claim representing the assumption on its containing Claim Refinement Model by labelling it “broken”. The transformer then propagates the “broken” label throughout the Claim Refinement Model using the label propagation algorithm described in Section 5.3. If a bottom-level claim on a Claim Refinement Model is labelled “broken”, the transformer makes one of the model modifications described in the previous section to the corresponding Strategic Rationale Model. After making a modification, adaptation policies need to be re-derived to take the model changes into account.

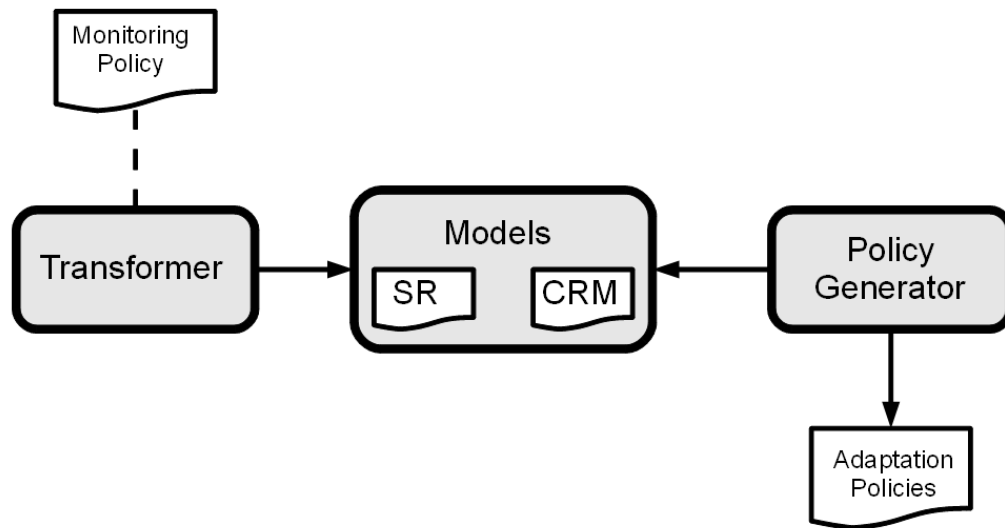
The specific model modification performed by the transformer in response to a monitoring event is controlled by a monitoring policy. Like adaptation policies, monitoring policies use an XML format, and specify the claim to be invalidated in response to a named monitoring event. The policy also stipulates which of the model modifications should be performed on the Level-One strategic rationale model should the “broken” label propagate to a bottom-level claim. Figure 40 depicts a snippet from a monitoring policy devised for the adaptive image viewer. The policy defines the model transformations that take place in response to the previously discussed “Caching will Improve Perceived Speed” claim being found invalid.

```
<MonitoringPolicy><MonitorRule>
  <Monitor><Event>event.CacheSlow</Event></Monitor>
  <Transformation>
    <Node type="claim">
      Caching will Improve Perceived Speed
    </Node>
    <Action>INVERT</Action>
  </Transformation>
</MonitorRule></MonitoringPolicy>
```

*Figure 40: Single Monitoring Rule from a Monitoring Policy*

In Figure 40 above, the “MonitoringPolicy” element serves as a container for one or more “MonitorRule” elements, which define a single model transformation. The “MonitorRule” element contains two child elements: “Monitor” and “Transformation”. The “Monitor” element specifies one or more “Event” elements, which stipulate the names of the events which signal the need to perform a model transformation. The “Transformation” element describes the model transformation required in response to the monitoring event; comprising of a “Node” element, which specifies the ReAssuRE model element upon which the modification is to be performed, and an “Action” element, which specifies which model modification is to be performed. Valid action types are “REMOVE”, “INVERT” and “EXCLUDE”; which correlate with the three model modifications described in Section 6.2. The implemented model transformer, which will be discussed shortly, performs the “EXCLUDE” modification upon  $i^*$  task elements in ReAssuRE Level-One Strategic Rationale models. With this implementation, the determination of which task a claim supports (and thus which to exclude from consideration in a decision) is made by the analyst at design time instead of by the m-DAS at run time when using the “EXCLUDE” transformation.

This thesis presents a proof-of-concept model transformer, capable of listening for monitoring events raised by assumption monitors and performing the model modifications specified by a monitoring policy. Model modifications that result in a bottom-level claim being invalidated result in adaptation policies being re-derived from the updated models. The model transformer loads monitoring policies in the format illustrated by Figure 40. a m-DAS utilising the proof-of-concept model transformer adopts the model modification and reasoning architecture depicted in Figure 41.



*Figure 41: Implemented Model Transformer Architecture*

In Figure 41, pairs of Level-One Strategic Rationale and Claim Refinement Models are reasoned with by the model transformer. Individual assumption monitors (devised externally) raise events, which the model transformer listens for. The monitoring policy in Figure 41 is loaded by the model transformer, controlling which model modifications are performed in response to which monitoring events. After model modification, a separate policy derivation sub-system uses the changed models to generate updated adaptation policies.

To conclude, this section has discussed the architecture by which a ReAssuRE-based m-DAS may be constructed. Monitors are devised for assumptions in claim refinement models, and in the event that monitoring data indicates an assumption is no longer valid the corresponding claim on the model is labelled “broken”. The “broken” label propagates throughout the Claim Refinement Model, and if it reaches a bottom-level claim, action is taken on the Strategic Rationale model. The precise action taken is defined in a monitoring policy, and performed by a model transformer. The section has introduced a



demonstrative model transformer capable of performing model modifications as specified in a monitoring policy.

## 6.4 Uncovering Emergent Behaviour in m-DASs

Every DAS risks displaying emergent behaviour as a result of uncertainty about the environment in which it operates, uncertainty in the suitability of available components to anticipated environmental conditions, and uncertainty as to how the DAS will respond to changes in the environment. A DAS may adopt a sub-optimal target system configuration in some circumstances, or adaptation itself may have some unforeseen consequence. Furthermore, the mis-identification of, or overlap between boundaries between domains can lead to a DAS performing frequent transitions between two or more target systems, a problem known as “thrashing”.

The additional autonomy granted a m-DAS, in terms of its ability to adapt to conditions outside those anticipated at design time, means that there is an even greater likelihood of encountering emergent behaviour. Unexpected combinations of components may be used, or one target system's configuration could, in whole or in part, be used outside its intended domain. For a DAS, it remains possible to guarantee that the system will always take the form of one of its specified target systems. a m-DAS offers no such guarantee, in that any of the target systems may have been modified.

Thus, the verification and validation of a m-DAS must ensure that emergent behaviour as a result of model modification is both not damaging, and compliant with specified behaviour in anticipated conditions. This section describes a modified version of the requirements validation scenario derivation method discussed in Section 5.3 to identify conditions in which a m-DAS may exhibit emergent behaviour, and to prioritise those scenarios that are considered most likely to occur. Once these scenarios are identified, assurance of a m-DAS's behaviour under certain sets of conditions can be delivered through testing.

In Section 5.3, claims were classified broadly by relative certainty, with less certain claims acting as markers for areas of uncertainty in developers' understanding of a domain or of expected behaviour. These areas of uncertainty could be explored using validation scenarios, seeking to eliminate the uncertainty, or at the very least to ensure that its consequences are not serious. In this section, a similar classification is performed, with less certain claims highlighting a need to test to ensure model modification that may occur will not yield damaging emergent behaviour.

Of course, testing is not the only means of validating software: for a DAS or m-DAS, it may be possible to deliver an acceptable degree of assurance through monitoring, or by allowing full or partial human control over adaptation. This latter suggestion remains an interesting and relatively unexplored research area, and there may well be scope for methods and software supporting human-instigated or human-approved adaptation to be developed in the future. However, such approaches do not offer complete assurance, and the assurance they do offer comes at the expense of system autonomy. Thus, this section focusses on identifying key scenarios to validate m-DAS behaviour through testing.

m-DAS testing activity can be separated into two categories of activity: testing of a m-DAS's business logic and of a m-DAS's adaptive behaviour. Testing of a m-DAS's business logic is akin to that carried out for traditional, non-adaptive systems. Testing a m-DAS's adaptive behaviour seeks to verify that the configurations adopted in given scenarios are optimal, or at the very least not damaging.

To test the adaptive behaviour of a conventional DAS, testing scenarios need to be designed for each of the target systems that can be selected, along with scenarios designed to test the DAS's ability to correctly identify which target system to adopt in each of the sets of expected operating conditions partitioned from the environment. This amounts to a considerable testing burden, but remains within the bounds of feasibility. Unfortunately, in order to fully test a m-DAS using assumption monitoring and model transformation as discussed in this chapter, it is necessary to identify and understand the m-DAS's scope for emergent behaviour completely. To offer this level of testing coverage, it would be necessary to devise scenarios for every possible combination of assumptions holding and

being broken. Each combination of assumptions holding or otherwise can be thought of as representing a testing scenario at the modelling stage.

If every possible combination of assumptions holding or otherwise needed to be tested to uncover emergent behaviour, the additional number of testing scenarios that would be required for each target system for a m-DAS over a standard DAS ( $T$ ) is yielded by the following formula:  $T=2^n-1$ . The  $n$  represents the number of bottom-level claims in the target system's Claim Refinement Model. Subtracting one removes the scenario in which all assumptions hold: the original target system design as tested in a standard DAS. This explosive testing burden means that offering complete assurance that a m-DAS will exhibit no emergent behaviour as a result of its model modification is infeasible for all but the simplest m-DASs.

In order to offer an acceptable degree of assurance against damaging emergent behaviour in m-DASS, some pragmatism is required: instead of devising testing scenarios for every possible combination of assumption holding or otherwise, test only those combinations most likely to occur whilst omitting those impossible or unlikely. There is, of course, a trade-off between the desire to minimise the number of testing scenarios devised (to minimise cost) and that of maximising assurance: devising testing scenarios for greater numbers of the less likely scenarios offers greater assurance at greater cost. The remainder of this section borrows (and adjusts) the claim classification method discussed in Section 5.3, to demonstrate the use of ReAssuRE models in pruning the testing space for m-DASs.

The method for identifying key requirements validation scenarios for DASs, discussed in Section 5.3, relies on applying a broad classification of relative claim certainty to underlying claims in Claim Refinement Models and labelling the underlying claims to reflect this classification. The applied labels are propagated throughout the model (and crucially, to bottom-level claims), and validation scenarios are devised to cover the least certain bottom-level claims to validate the assumptions on which they are based, or to verify that the consequences of the assumptions no longer holding were not damaging. Claims are classified in terms of certainty as either: unbreakable, qualified, or uncertain.

An unbreakable claim is axiomatic, whilst an uncertain claim is one in which a low degree of confidence is held. A qualified claim is considered relatively safe, and fall between the two extremes of unbreakable and uncertain claims.

When using this categorisation to prioritise testing scenarios aiming to uncover emergent behaviour in a m-DAS, a slight change to the definition of the “Unbreakable” category is required. In this context, claims that the m-DAS has no means of monitoring directly *and can have no other certainty label propagated to them* are considered unbreakable. Claims that are monitored may be considered only qualified or uncertain. By inference, an axiomatic claim need not have a monitor devised for it. A claim that is unmonitable and has no possibility of a broken label being propagated to it has no scope for causing emergent behaviour through its invalidation, and can as such be removed from consideration.

Returning to the adaptive image viewer, Figure 22, in Section 5.1.2 on page 90, depicted a revised design for the  $S_2$  target system. A revised design is used because the original  $S_2$  target system for the adaptive image viewer has just one bottom-level claim, which is of little illustrative benefit here.

The revised  $S_2$  target system has a claim stating that the originally specified cache is ineffective in  $S_2$ , along with two claims supporting the selection of the newly devised learning cache, which uses simple machine learning to predict images likely to be loaded soon in order to cache them. Figure 21 on page 88 depicted the Claim Refinement Model supporting the three bottom-level claims on Figure 22.

In Chapter 5, this revised  $S_2$  target system design and supporting reasoning were arrived at thanks to the cache having been proven ineffective in use. Given that initial understanding of the domain and the behaviour of DAS components was inaccurate (the analyst failed to predict that images would be accessed non-sequentially and thus the cache prove ineffective), it is reasonable to assume that this new design and reasoning may too be flawed. Thus, there is some benefit to converting the adaptive image viewer to a m-DAS, which should prove better equipped to adapt to any further unforeseen

circumstances. By monitoring some of the assumptions in the claim refinement model, the adaptive image viewer should be able to adjust the design of the  $S_2$  target system if the assumptions upon which it is reliant are again proven false.

There are two claims on Figure 21 that are modelled with a “broken” label. Although such claims don't influence a target system design, a monitor could be devised to remove the broken label should the assumption be found to hold in the future. If this were to occur, the bottom level-claim “Extra Speed Needed in  $S_2$ ” would be reinstated in the Strategic Rationale model (not shown), and potentially cause the m-DAS to change behaviour. When applying confidence labels to these initially broken claims, the analyst is essentially expressing a degree of confidence that the claim will remain broken, rather than a degree of confidence that the claim will hold at some time in the future. Figure 42 below shows the claim refinement model from Figure 21 annotated with appropriate confidence labels.

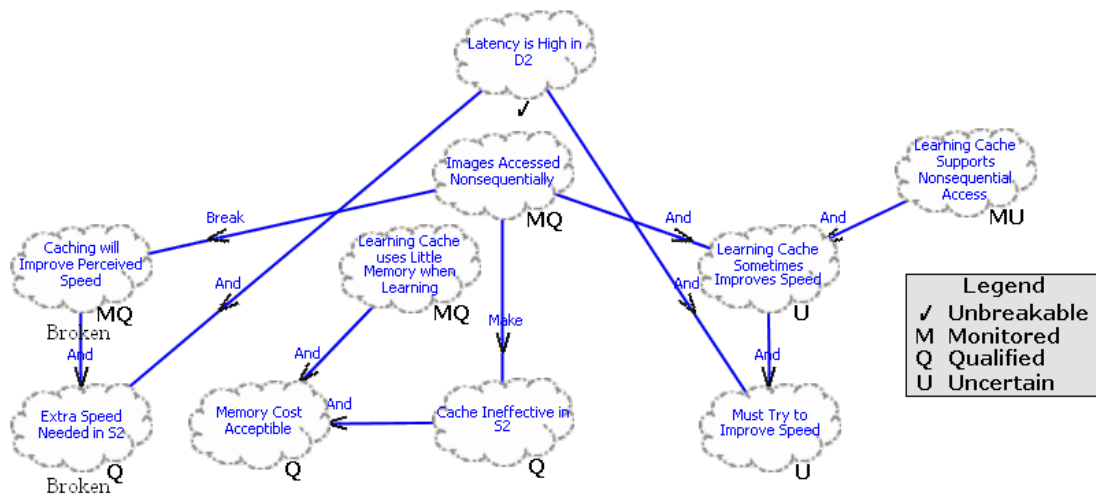


Figure 42: Annotated Claim Refinement Model for Learning Cache

As Figure 42 depicts, only the “Latency is High in  $S_2$ ” claim has been categorised as unbreakable: although it would certainly be feasible to devise a monitor for the claim, the fact that the defining characteristic of the  $D_2$  domain is high latency means that monitoring it is unnecessary. There are four claims for which monitors may be devised,

and the certainty labels from these claims are propagated throughout the claim refinement model in accordance with the rules set out in Section 5.3. Thus, for the revised  $S_2$  target system, there are three bottom-level claims considered qualified, and one considered uncertain.

Combinations of uncertain bottom-level claims are the key focus of testing aimed at uncovering emergent behaviour in a m-DAS as a result of model modification. Ideally all combinations of bottom-level claims not characterised as unbreakable would have tests devised for them. However, in circumstances in which devising so many test cases proves infeasible, focussing on the m-DAS's behaviour should one or more uncertain bottom-level claims no longer hold, will yield some degree of assurance whilst minimising testing burden. In this example, the behaviour of the m-DAS should the “Must Try to Improve Speed” claim no longer hold emerges as the key testing focus for uncovering emergent behaviour. Table 4 below uses the  $T=2^n-1$  formula discussed previously to illustrate the number of testing scenarios required to test m-DAS behaviour after invalidating combinations of all claims in Figure 42, qualified & uncertain claims, or just uncertain claims at the bottom level.

All Claims	Qualified and Uncertain	Uncertain
15	15	1

*Table 4: Testing Scenarios Required to Cover the Categories of Claims*

In this instance, because there are no unbreakable bottom-level claims, testing the m-DAS's behaviour after invalidating combinations of qualified and uncertain claims is an identical task to testing all combinations of claims. If only uncertain bottom-level claims are considered, a single testing scenario is required; highlighting the m-DAS's behaviour when this particular claim is invalidated as a key testing concern. Of course, any procedure reliant on a qualitative certainty classification is vulnerable to the problem of mis-classification; a claim classified as qualified that later proves false may still cause emergent behaviour after evading testing coverage if only uncertain claims were

considered. It is, therefore, advisable to cover all but the unbreakable claims with testing in m-DASs where a risk of emergent behaviour after model modification is unacceptable.

The testing approach advocated in this section offers no coverage of the behaviour of individual components, or of the system's business logic in any individual configuration. System components are treated in a black-box manner. However, in terms of potential modification and resultant changes to the m-DAS's adaptive behaviour, the approach is analogous to a testing approach that would offer path coverage of a given piece of code. White-box analysis of Claim Refinement Models allows paths of “Broken” label propagation to be identified and grouped, with analysis of the SR models allowing all potential paths the of model transformer, in light of the results of the “Broken” label propagation analysis, to be investigated. Of course, testing only combinations of qualified and uncertain claims, or solely combinations of uncertain claims reduces testing coverage accordingly, with this being analogous to testing the most likely execution paths when performing standard white-box software testing.

To conclude, this section has explored the issue of emergent behaviour in m-DASs. It has identified a difficulty in delivering assurance in the potentially modified behaviour a m-DAS may exhibit after model modification through testing. The testing workload required to offer a good level of assurance in a m-DAS's behaviour in potentially unforeseen or only partially understood circumstances is so burdensome as to often prove infeasible. To combat this, the chapter has introduced a method by which limited testing resources can be directed to the most likely of scenarios potentially uncovering emergent behaviour, using ReAssuRE models to remove the need to perform complex analysis to establish the relative certainty of complex claims by deriving the relative certainty of simpler, easier to categorise claims. The issue of testing complexity for m-DASs, and the test-case pruning method discussed in the chapter have been published in preparation for this thesis [4].

## 6.5 Chapter Conclusion

Previous work in requirements monitoring has suggested that systems may benefit from monitoring the degree to which they are able to satisfy certain key requirements. A DAS, however, has the potential to take action in circumstances where this monitoring data indicates under-performance or a failure to satisfy requirements. Also, the emerging *models@run.time* paradigm in which a system loads and reasons with models of its own behaviour indicates an interest in this style of system. This thesis uses the term model-driven Dynamically Adaptive System (m-DAS) to identify a DAS that uses its design-time models to guide its run-time adaptation, and this class of system represents a step forward in both of the aforementioned research areas.

This chapter illustrates how a m-DAS could be constructed using ReAssuRE models to guide run-time adaptation, with monitoring of assumptions used as a means of modifying models to reflect unanticipated operating conditions. The chapter discusses the different components necessary to build such a system, and presents a reference implementation of the key model transformer component capable of modifying models in response to monitoring data (in the form of events) in accordance with a monitoring policy. The potential increase in autonomy offered by a m-DAS comes at a price of complexity, and thus a m-DAS offers less predictability than a DAS. The chapter explores the issue of assurance in m-DASs, with the reduction in predictability remaining challenging.

The chapter also provides answers to two of this thesis' research questions, which are presented in Section 1.3:

How can a system be designed with a greater degree of autonomy than current state-of-the-art DASs, and is the extra autonomy useful?

a m-DAS using ReAssuRE models to control adaptation and assumption monitoring to adjust the models as the environment change offers a greater degree of autonomy than a current state-of-the-art DAS, in that the m-DAS possesses a limited ability to adjust its behaviour to suit operating conditions *outside* those envisaged at design time. This ability



offers the possibility of designing systems to operate in environments with a greater degree of uncertainty than previously possible, whilst still maintaining the ability of a DAS to operate in volatile, changeable environments.

How can the testing workload be managed in systems with greater autonomy?

The answer to this question is less positive: the testing workload in a m-DAS using ReAssuRE models and assumption monitoring is high. It is necessary, as it is for standard DASs, to test individual target systems, and the mechanism that decides on and performs the transitions between them. Additionally, it is necessary to verify that the m-DAS's behaviour should an assumption no longer hold is correct, or at the very least not damaging. It is possible to afford some degree of assurance through testing by prioritising the most likely scenarios in which emergent behaviour may occur, and it is possible to remove scenarios that have no possibility of provoking emergent behaviour from the testing regime. However, in many such m-DASs a degree of assurance will have to be sacrificed to keep testing workloads feasible. For m-DASs in which this sacrifice is unacceptable, other methods of delivering assurance such as monitoring or human-in-the-loop adaptation may be worthwhile, but are beyond the scope of this thesis.

The next chapter seeks to validate the work presented in this and the previous chapter by way of a more complete case study, which focusses on a more complex DAS than has been used to illustrate the approach.

## 7 Evaluation

---

This chapter validates the ReAssuRE modelling approach's usefulness for larger, more complex DASs than the adaptive image viewer used as an example throughout this thesis, by way of a retrospective case study, focussing on a previously deployed DAS. The novel model uses demonstrated throughout Chapters 5 and 6 are validated in this larger case study, demonstrating the benefits discussed in the respective chapters on a real-life DAS.

GridStix is a DAS deployed on the River Ribble in North West England that performs flood monitoring and prediction [2]. It takes the form of an intelligent wireless sensor network with multiple nodes measuring river depth and flow rate using a variety of sensors, including the analysis of images taken with an on-board digital camera. The GridStix system uses the GridKit middleware [39], which provides the system's adaptive capabilities.

The flow rate and river depth data is used by a point prediction model, which predicts the likelihood of the river flooding using data from the local node and data cascaded from nodes further upstream. The more upstream data available, the more accurate the prediction.

The environment in which GridStix operates is volatile, as the river is liable to flooding. When the river floods, the nodes are in danger of submersion, and of sustaining significant damage from water-borne debris. As such, the GridStix system needs to be able to maximize its ability to withstand node failure in order to maintain the connectivity of the surviving nodes.

The GridStix nodes have processing capability, which allows processing of the data and execution of flood prediction models on-site with the GridStix system acting as a lightweight grid. However, the nodes are resource-constrained and some tasks, particularly flow rate calculations which involve processing digital camera images, are best performed by distributing computation among the nodes which increases prediction accuracy.

Distributing computation has a cost in terms of the power consumed by inter-node communication, which is a serious issue since GridStix's location is remote, and power has to be provided by batteries and solar panels. The GridStix system can communicate using different spanning tree algorithms, trading off fault tolerance and power consumption. As such, even from this brief overview it is possible to infer that GridStix has three key conflicting softgoals: “Energy Efficiency”, “Fault Tolerance” and “Prediction Accuracy”.

When specifying the GridStix system, domain experts (hydrologists) partitioned the operating environment into three distinct domains:  $D_1$  (Normal),  $D_2$  (High Flow) and  $D_3$  (Flood). The  $D_1$  (Normal) domain is characterised by a quiescent river, with little imminent risk of flood or danger to local residents or the nodes themselves. The  $D_2$  (High Flow) domain features a fast-flowing river, that may suddenly flood. The  $D_3$  (Flood) domain occurs when the depth increases and the river is about to flood, which means that the nodes are in imminent danger of failure. For each domain, a target system was devised, with target system  $S_1$  tailored for domain  $D_1$ ,  $S_2$  for  $D_2$  and  $S_3$  for  $D_3$  respectively.

The GridStix developers whilst prototyping constructed a basic simulator [113], capable of emulating inter-node communication, visualising the currently active network topology of the GridStix nodes under simulation, and calculating both the power used by node operation and the power generated by the nodes' solar panels. To validate the behaviour of GridStix system under configurations devised using ReAssuRE models (either automatically or manually) for scenarios discussed in this chapter, several features have been added.

Firstly, the simulator was extended to support the loading of adaptation policies directly, rather than a simulator scenario needing to be coded to set up the simulated GridStix nodes in a given configuration.

Secondly, support has been added for scripting simulated weather conditions, which affects the behaviour of the river (and thus the behaviour of the GridStix nodes, as they observe the river) and the performance of the GridStix nodes' solar panels.

Finally, support has been added allowing node failures to be simulated: the GridStix system's operating environment is hostile, with nodes potentially failing due to debris in the flooding river, prolonged exposure to notably cold and wet conditions, or even due to interference from local wildlife.

Together, these changes allow prototype adaptation policies to be analysed before being deployed on the actual system. The support for scripting weather conditions allows the performance of the GridStix nodes in various configurations to be assessed in both conditions previously experienced by the GridStix system and recorded, and conditions specially designed to test the configuration under study to the extreme. A need to better understand and optimise the behaviour of the GridStix system after losing one or more nodes to damage emerged after deployment, a task eased by the simulator's support for forcing node failure.

The results of a simulator run for a given configuration under a given set of environmental conditions are typically expressed quantitatively by way of the number of simulated hours elapsed before a sufficient number of GridStix nodes fail to render the network fragmented i.e. one or more nodes unable to send or receive data to or from the rest of the nodes. A fragmented network does not necessarily represent a complete system failure, the remaining body of nodes able to communicate will still work together to reach flood predictions, but the accuracy of the predictions will be compromised by the non-availability of data from serviceable nodes. Other less frequently used metrics are the number of GridStix nodes that either fail due either to battery exhaustion or become fragmented from the network over a set number of hours, or the number of hours elapsed before only a set number of GridStix nodes remain in communication with each other and able to arrive at a flood prediction.

In addition to these quantitative results, the visualiser enables certain other observations to be made during simulation runs, which are more qualitative in nature. Observing the simulated GridStix system transition between target systems repeatedly, a condition known as thrashing, would indicate a problem with the trigger conditions controlling adaptation, and ultimately a problem with the domain partitioning carried out

during initial analysis. Observing that in a given scenario, one specific GridStix node seems to be using significantly more power than the others, for example because the chosen network topology routes a significant proportion of traffic through the node, may allude to a potential node failure if the scenario is elongated, or if more challenging environmental conditions are encountered. These observations do not form part of the simulator results, but offer a useful means of feedback in both refining configurations and designing challenging weather condition scripts for simulation.

## 7.1 ReAssuRE models for the GridStix System

This section presents the complete set of ReAssuRE models for the GridStix system, discussing each in turn. At the highest level of abstraction, a Level-One Strategic Dependency model is created for the GridStix system as a whole, showing the actors, goals and dependencies involved in the system. This SD model is depicted in Figure 43.

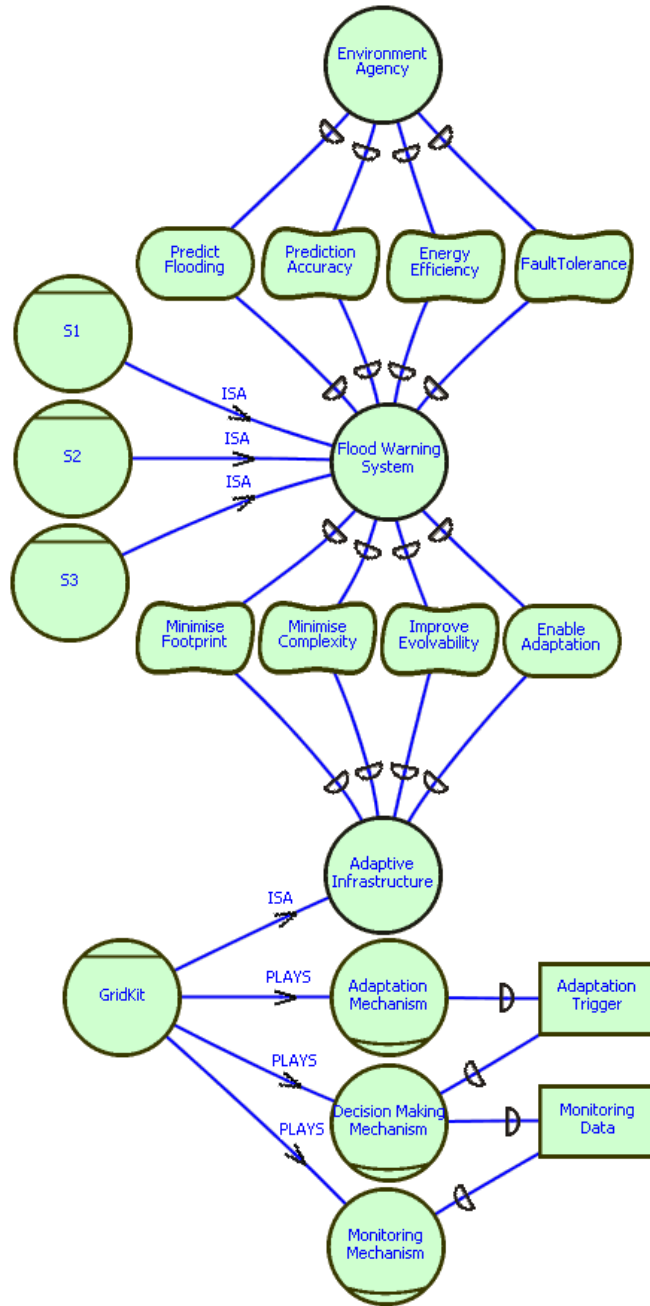


Figure 43: Level-One Strategic Dependency Model for GridStix System

The Strategic Dependency model for the GridStix system depicted in Figure 43 is considerably more complex than that created for the adaptive image viewer (depicted in Figure 2 on page 50). The GridStix system's dependency on the GridKit middleware means

an additional actor on the model, and additional dependency relationships to reflect the dependencies between the GridStix system and its adaptive infrastructure. Figure 43 shows that the principal stakeholder for the GridStix system is the UK's Environment Agency, who are responsible for managing and monitoring the river. The Environment Agency depend upon the GridStix system to predict flooding, and to do so as accurately and reliably as possible whilst using as little energy as possible.

It could be argued that the need for “Energy Efficiency” and “Fault Tolerance” are both derived from a higher-level reliability softgoal. However, the GridStix system's environmental constraints prevent any higher-level softgoal from being satisfied (sufficiently satisfied) via means other than the system having a degree of fault tolerance and managing power consumption to prevent battery exhaustion. Thus, the higher-level softgoal remains out of scope in Figure 43.

The only difference between the GridStix Level-One SD model depicted in Figure 43 and a LoREM Level-One SD model of the DAS is the separation of GridKit's roles in GridStix. As discussed in Section 4.3, The roles of an adaptation mechanism, a decision-making mechanism and an adaptation mechanism are separated from the adaptive infrastructure role in the ReAssuRE modelling process to allow for DASs in which the three roles are played by separate concrete agents, and to provide more detail at Level Two. The three roles are likewise separated in the SD model, but in systems such as GridStix in which the three roles are played by a single agent, the extra detail has no specific additional benefit.

### 7.1.1 Level-One Strategic Rationale Models

As discussed in the chapter introduction, the GridStix system's operating environment was partitioned by domain experts into three domains: normal, high flow and flood; referred to as  $D_1$ ,  $D_2$  and  $D_3$  respectively. The target systems  $S_1$ ,  $S_2$  and  $S_3$  correspond to

each domain, and this sub-section presents and discusses the Level-One Strategic Rationale models for each in turn.

Figure 44 depicts the Level-One strategic rationale model for the  $S_1$  target system.

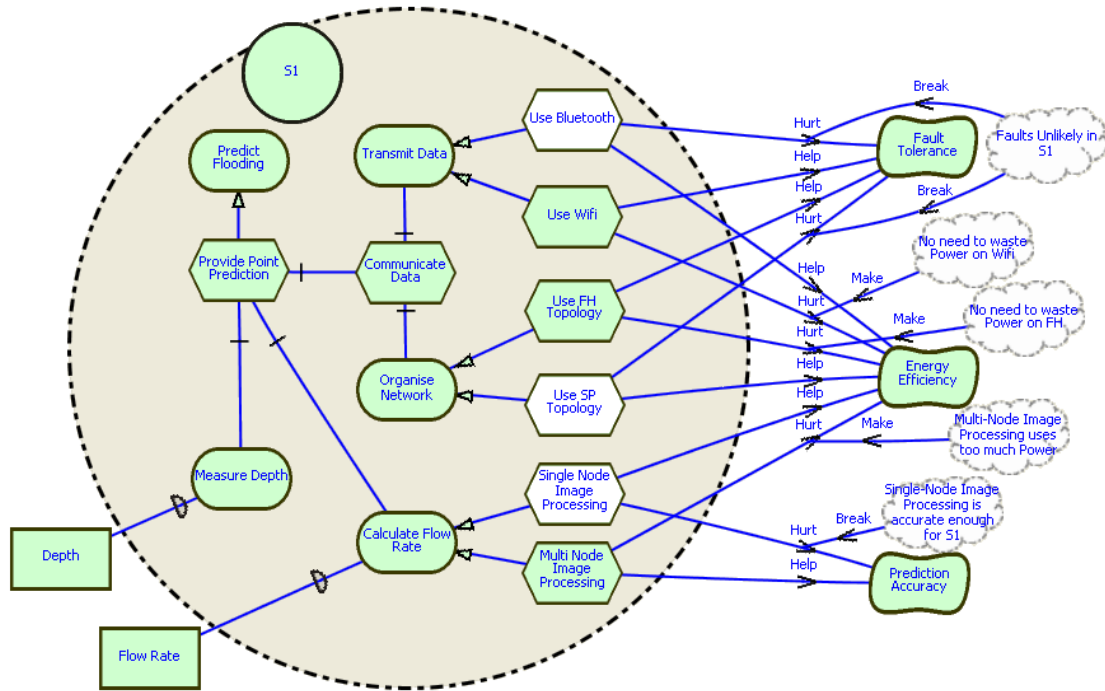


Figure 44: Level 1 SR Model of GridStix  $S_1$  Target System

Figure 44 shows that the GridStix system's main goal (upon which the stakeholders depend on the GridStix system fulfilling) is to “predict flooding”. The GridStix system opts to fulfil this goal using a point prediction algorithm, as indicated by the “provide point prediction” task. The task of providing a point prediction is complex, and is divided into: “measure [river] depth”, “calculate [river] flow rate” and “communicate data”. Measuring the depth and flow rate depends of course on the data being available from sensors, which remain out of scope on Figure 44. The “communicate data” task may be completed by satisfying the “organise network” and “transmit data” goals. “Organise network” may be satisfied either by organising the GridStix Nodes using a fewest hop or shortest path



topology, as controlled by the “use SP topology” and “use FH topology” tasks, respectively. The “transmit data” goal may be achieved by using either Bluetooth or Wi-Fi for radio transmission, as indicated by the “use Bluetooth” and “use Wifi” tasks respectively. Finally, the “Calculate Flow Rate” Goal may be achieved either by analysing images taken with the on-board digital camera on the local node, or by distributing the calculation throughout the grid, as suggested by the “Single Node Image Processing” and “Multi Node Image Processing” tasks, respectively.

From an adaptation perspective, there are three variation points in Figure 44, where a goal may be satisfied by two or more tasks to be decided between. These goals are “Transmit Data”, “Organise Network” and “Calculate Flow Rate”. For the “Transmit Data” goal, “Use Bluetooth” hurts the GridStix system's fault tolerance, because its inferior range means that fewer nodes are within transmission distance of one-another. However, using Bluetooth to transmit data uses significantly less power than Wi-Fi. For the “Organise Network” goal, organising the GridStix nodes using a fewest hop topology increases the fault tolerance of the GridStix system as fewer nodes are depended upon to relay a single message, but does so at a cost of greater power consumption, given that each hop's transmission takes place over a greater distance. For the “Calculate Flow Rate” goal, using single-node image processing saves a significant amount of power by reducing the need for radio transmission, but the (significant) length of time taken to analyse the image and yield a flow rate value on a single node means that point predictions can be run less frequently, which has the effect of worsening the accuracy of flood predictions.

The  $D_1$  domain is characterised by a quiescent river, with little risk of either the river flooding or the GridStix nodes becoming damaged. As such, the  $S_1$  target system seeks to conserve power wherever possible, maximising energy efficiency. If possible, the GridStix system's stakeholders would prefer to offer the most accurate predictions and the greatest degree of fault tolerance possible, but the need to conserve energy (and thus charge the battery where possible whilst in this configuration) dictates a more balanced approach.

The rationale behind each selection decision is recorded in the five claims on Figure 44. The decision to use Bluetooth over Wi-Fi for data transmission is supported by two

claims. The “Faults Unlikely in  $S_1$ ” claim essentially downplays the significance of the negative contribution using Bluetooth makes to fault tolerance. The “No Need to Waste Power on Wi-Fi” claim speaks to the fact that, given that the river is quiescent and the risk of node failure due to damage is diminished, increasing power consumption to increase resilience is unjustifiable, highlighting the negative impact of “Use Wifi” on energy efficiency.

The decision to organise the GridStix nodes using a shortest path topology instead of fewest hop, like the previous decision, is supported by the “Faults Unlikely in  $S_1$ ” claim. The claim's “break” contribution argues that the negative impact of “Use SP Topology” on “Fault Tolerance” should be overlooked. The “No Need to Waste Power on FH” argues that the additional power required by a fewest hop topology to improve resilience is not justified, much like the “No Need to Waste Power on Wi-Fi” claim in the previous paragraph.

The final decision, governing the use of single node image processing over multi node, is supported by the “Multi-Node Image Processing uses Too Much Power” claim and the “Single Node Image Processing is Accurate Enough for  $S_1$ ” claim. The negative impact of “Single Node Image Processing” on the “prediction accuracy” softgoal is negated by the former claim, and the latter claim highlights the increased power consumption of “Multi Node Image Processing”.

Moving on to the  $D_2$  domain, the GridStix system is here faced with a river whose flow rate has increased, and is in danger of flooding relatively suddenly. In these circumstances, it is particularly important that the  $S_2$  target system is specified such that flood predictions are as accurate and timely as possible. Furthermore, should the river begin to flood unexpectedly, there is a danger of node failure. Figure 45 depicts the Level-One strategic rationale model for the  $S_2$  target system.

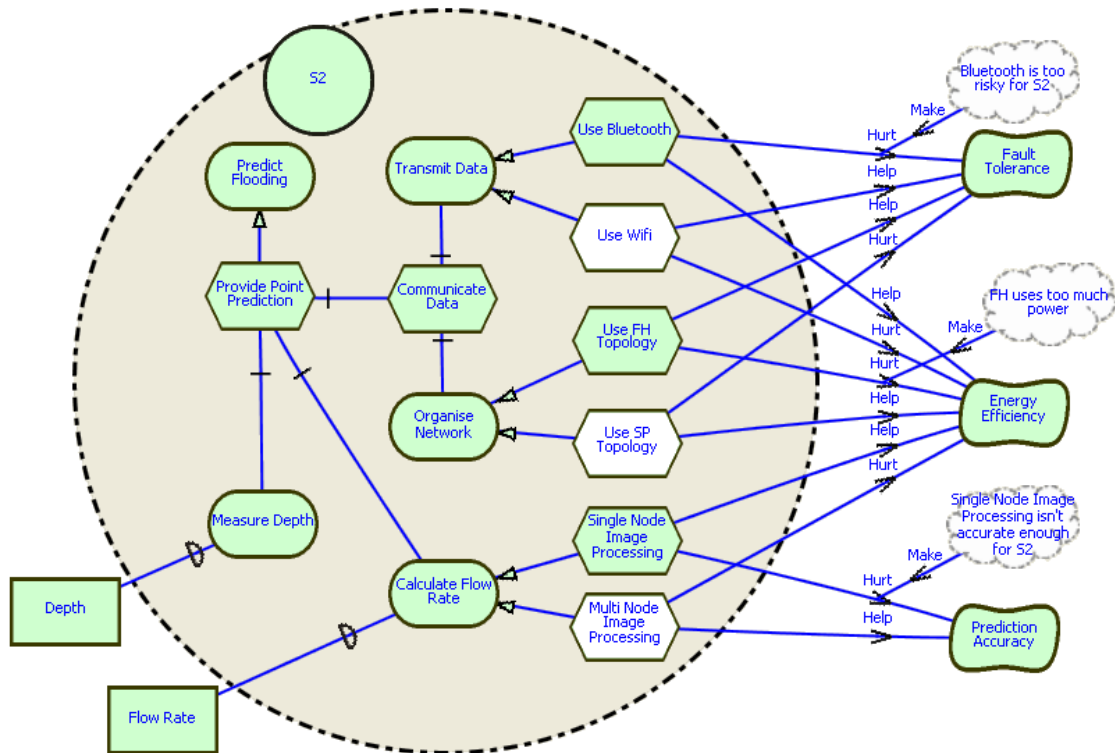


Figure 45: Level 1 SR Model of GridStix  $S_2$  Target System

The  $S_2$  target system has the same goals and means to achieve them as the  $S_1$  target system, along with the same variation points. However, the decisions reached on the specification of the  $S_2$  target systems are different, given the different circumstances in which it operates. The three claims in Figure 45 give a brief indication of the rationale behind each decision.

The decision to use Wi-Fi rather than Bluetooth for communication in  $S_2$  is supported by the “Bluetooth is too risky for  $S_2$ ” claim, which alludes to the fact that using Bluetooth with its shorter range would risk network fragmentation should a relatively small number of nodes fail. The claim’s “make” link highlights the negative impact selecting Bluetooth would have on the “Fault Tolerance” softgoal.

Likewise, the decision to use multi-node image processing to determine the river’s flow rate from images taken with nodes’ on board digital cameras, rather than performing

the analysis on a single node is supported by the “Single Node Image Processing Isn't Accurate Enough for  $S_2$ ” claim. The rationale here is that the additional time taken to analyse images on a single node means that predictions are made less frequently and with less data, reducing the accuracy. As with the previous claim, the “make” link highlights the negative impact of the “Single Node Image Processing” task on the “Prediction Accuracy” softgoal.

The final decision, to use a shortest path topology to organise the nodes on the network, has been reached on power-saving grounds, much as it was in the  $S_1$  target system. This time, however, the rationale is simpler. For  $S_1$ , a fewest hop topology was adjudged not to be necessary, as well as being expensive in terms of energy. For  $S_2$ , the risk of sudden flood and potential node failure means that any measure improving resilience would be welcome, but the additional power consumption associated with a fewest hop topology simply could not be afforded. As such, for the  $S_2$  target system a simpler rationale is recorded with the single “FH uses too much power” claim, which uses a “make” link to highlight the negative impact “Use FH Topology” would have on the “Energy Efficiency” softgoal.

The third,  $D_3$  domain, is characterised by a river that has begun to flood. In these circumstances the GridStix nodes are in significant danger of submersion which would (hopefully only) temporarily prevent them from communicating with the other nodes or collecting solar power. There is an additional danger of a node sustaining damage from debris floating downstream. As such, the  $S_3$  target system's specification focusses on resilience. Figure 46 below depicts the Level-One strategic rationale model for the  $S_3$  target system.

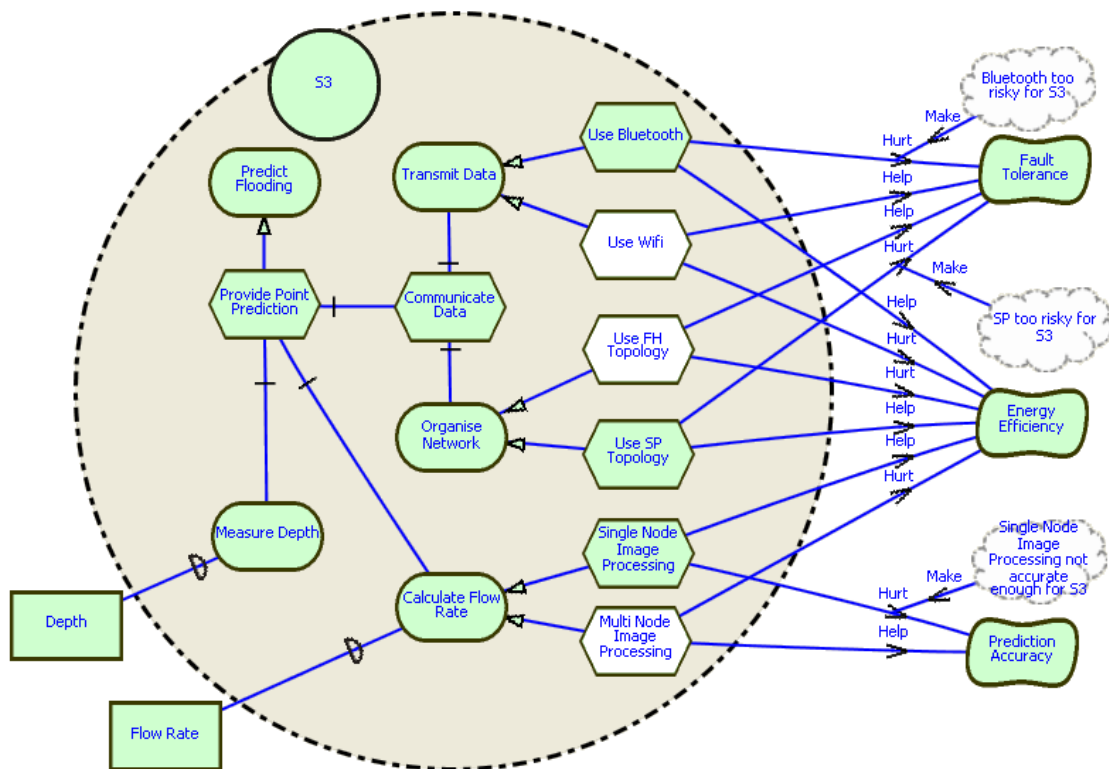


Figure 46: Level-One SR model for GridStix  $S_3$  Target System

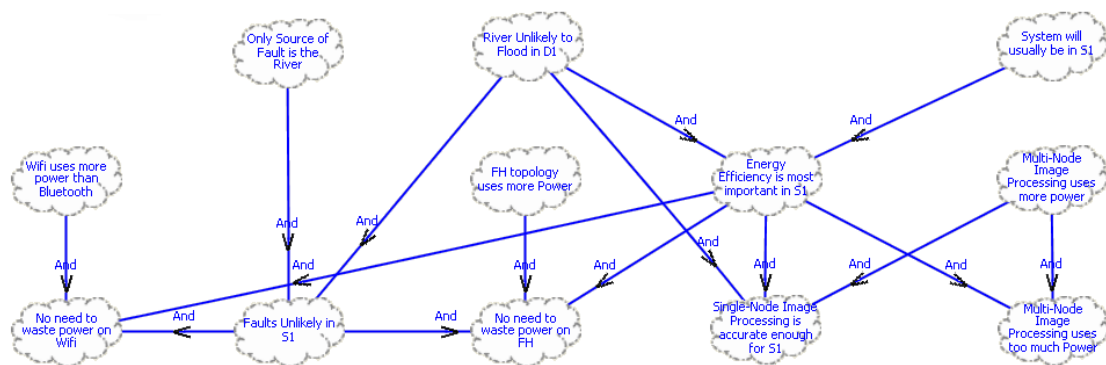
The three claims in Figure 46 record the rationale behind the selection decision taken for each variation point. The decision to use Wi-Fi for communication rather than Bluetooth, and the decision to use multi-node image processing rather than single-node share the same outcome and short rationale with the  $S_2$  target system. The third decision, which dictated the use of a fewest hop network topology instead of shortest path, is where the  $S_3$  target system design differs to that of  $S_2$ . For this decision, it was decided that the significant risk of nodes becoming temporarily or permanently unable to form part of the GridStix network meant that a shortest path topology, despite its lower energy consumption, should be overlooked in favour of the more resilient fewest hop topology. The “SP is too risky for  $S_3$ ” claim uses a “make” link to emphasise the negative contribution “Use SP Topology” makes to the “Fault Tolerance” softgoal.

The Level-One SR models presented in this subsection are significantly more complex than their LoREM counterparts, which present only the selected alternative for each decision in a target system, and contain no claims. The extra complexity improves the traceability of the ReAssuRE models significantly, as detailed in Section 7.2.

### 7.1.2 Level-One Claim Refinement Models

The previous sub-section introduced the Level-One Strategic Rationale models for each of the target systems specified for GridStix. This sub-section focusses on the Level-One Claim Refinement Models, which elaborate on the brief rationale recorded by the claims on the Level-One Strategic Rationale models presented and discussed in the previous sub-section. The claim refinement models for the  $S_1$ ,  $S_2$  and  $S_3$  target systems are discussed, in turn, below.

Figure 47 depicts the Level-One Claim Refinement Model for the  $S_1$  target system.



*Figure 47: Claim Refinement Model for GridStix  $S_1$  Target System*

The five claims on the Level-One Strategic Rationale model depicted in Figure 44 on page 141 are repeated in Figure 47 as bottom-level claims. These are supported, in various combinations, by four claims made about the environment, and three about the behaviour of various components available for selection. The four environmental claims: “Only

Source of Fault is the River”, “River is Unlikely to Flood in  $S_1$ ”, “System will Usually be in  $S_1$ ” and “Energy Efficiency is Most Important in  $S_1$ ” are relatively self-explanatory. The “Only Source of Fault is the River” claim was added when revisiting the model, because the “River Unlikely to Flood in  $S_1$ ” had previously been the sole support for the “Faults Unlikely in  $S_1$ ” claim. The fact that the river is unlikely to flood is visibly insufficient to make the derived assertion: which indicated an implicit assumption. Thus, the previously un-acknowledged “Only Source of Fault is the River” assumption is made explicit and modelled.

The three claims about the behaviour of available components: “Wi-Fi Uses More Power than Bluetooth”, “FH Topology uses More Power [Than SP]” and “Multi-Node Image Processing uses More Power [Than Single-Node]” are explicit statements of facts underpinning the rationale in the bottom-level claims. Should one of these claims later be discovered to be incorrect, the decisions made using this faulty understanding can be revisited.

Figure 48 depicts the Level-One Claim Refinement Model for the  $S_2$  target system.

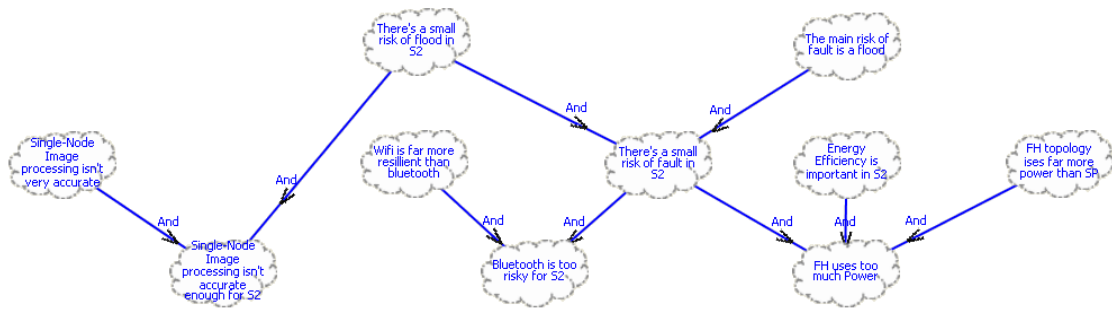


Figure 48: Claim Refinement Model for GridStix  $S_2$  Target System

As with the  $S_1$  target system, the three claims from the strategic rationale model depicted in Figure 45 on page 144 appear in Figure 48 as bottom-level claims. The first: “Single Node Image Processing isn't accurate enough for  $S_2$ ”, is supported by two underlying claims: “There's a small risk of flood in  $S_2$ ” and “Single Node Image Processing isn't very accurate”. Together, these two supporting claims elaborate on the rationale in

the bottom level claim: that, given the risk of flooding whilst the GridStix system is in the  $S_2$  configuration, the lesser degree of accuracy consequent to performing image flow rate analysis on a single node is unacceptable.

The remaining two bottom-level claims are particularly interesting, because the softgoal contributions of the candidate tasks on the associated Strategic Rationale model are identical yet the decisions reached in each case are conflicting. These conflicting decision outcomes are one of the motivators for the use of claims over some quantitative contribution link or softgoal weighting scheme to record decision rationale. In the case of the decision governing data transmission method, the more power hungry and more resilient Wi-Fi is used, whereas for the decision governing network topology, the less resilient and more energy efficient shortest path topology is used. For the former, the rationale captured by the claims is that the risk of flooding in  $D_2$  implies a risk of node failure, and that Wi-fi transmission improves resilience considerably. For the latter, the rationale is that the slight improvement in resilience offered is overridden by a dramatic increase in power consumption (given that radio transmissions take place over increased distances using considerably more power).

Figure 49 depicts the Level-One Claim Refinement Model for the  $S_3$  target system.

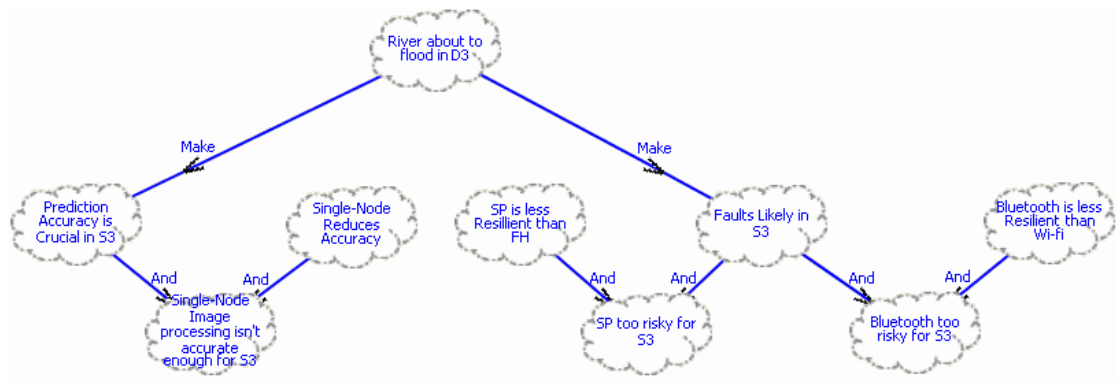


Figure 49: Claim Refinement Model for GridStix  $S_3$  Target System



As with the previous two target systems, the three bottom level claims on the associated strategic rationale model (Figure 46 on page 146) appear in Figure 49 as bottom-level claims. Unlike the  $S_2$  target system, however, the decisions reached for the  $S_3$  target system are consistent with one-another. In  $S_3$ , the imminent risk of flood and node failure means that the system adopts its most accurate and most resilient configuration at the expense of energy efficiency. The rationale for all three decisions is that given the risk of flood and therefore fault, the more resilient or accurate alternative should be preferred.

Although the additional tracing information recorded on ReAssuRE Level-One SR models allows far more rationale inferred than their LoREM counterparts, or any standard  $i^*$  model, the rationale it is possible to convey in a short claim label is often imprecise or simplistic. The ability to record a more detailed rationale using Claim Refinement Models serves not only for later reference, but allows the basis of a decision rationale to be examined more thoroughly.

The case study has demonstrated that merely the act of constructing a Claim Refinement Model can lead to previously implicit assumptions being identified. An unidentified assumption underpinning potentially several decisions could, potentially, require significant changes to a DAS's requirements specification should it later be found false. Uncovering the assumption earlier in the requirements engineering process allows the assumption's safety to be assessed whilst the consequences of adjusting design decisions based upon it are less severe. In the case of the uncovered assumption in the GridStix system, this meant that the possibility of external interference to the system by local wildlife was considered. As a result, extra attention was paid to decisions affecting GridStix's fault tolerance, and that simulation was undertaken to establish the potential impact of node failure caused by forces other than the river.

### 7.1.3 Level Two Models

The previous two sub-sections presented the different types of Level-One model created for individual target systems during the ReAssuRE process. Level-Two models focus on the transitions between target systems, defining the condition(s) triggering adaptation and pairs of target systems which can be validly transitioned between. For the GridStix system, all three target systems may transition to any of the others validly, which would require six Level-Two models. However, given the high degree of similarity between the models only a single Level-Two model will be presented and discussed in this sub-section. Figure 50 shows the Level-Two model covering the transition between the  $S_1$  and  $S_2$  target systems.

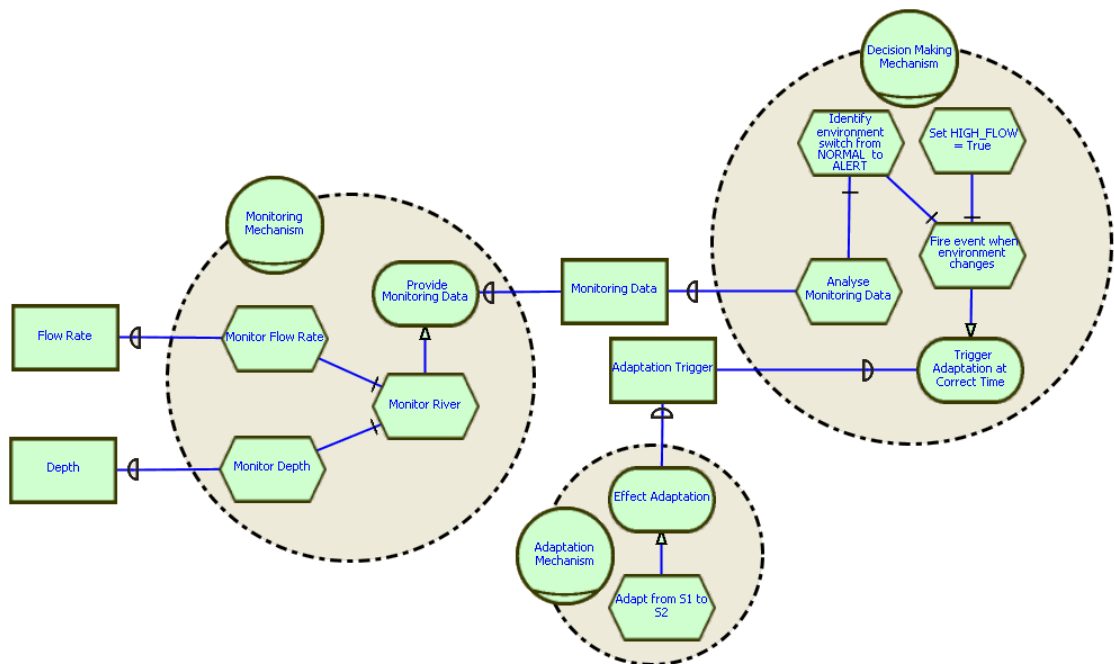


Figure 50: Level 2 Model of GridStix  $S_1$  to  $S_2$  Transition

As in the Image Viewer Level Two model discussed in Section 4.3, Figure 50 is separated into three distinct roles: the monitoring mechanism, decision making mechanism and the adaptation mechanism. In the GridStix system, the GridKit adaptive

middleware [39] plays each of the roles, but this will not necessarily be the case for other DASs. The monitoring mechanism monitors the flow rate and depth of the river, collating and supplying the data to the decision making mechanism. The decision making mechanism determines the current state of the environment, identifying a change of domain from  $D_1$  to  $D_2$ . When this occurs, the decision making mechanism triggers adaptation, in this case by setting the “HIGH\_FLOW” flag, which is effected by the adaptation mechanism which transitions the GridStix system from  $S_1$  to  $S_2$ .

As discussed in Section 7.1, the additional complexity in ReAssuRE models stemming from the separation of LoREM's “Adaptive Infrastructure” role into the three roles depicted in Figure 50 is of limited illustrative benefit in systems, such as GridStix, in which all three roles are played by a single agent. In systems utilising an adaptive infrastructure that plays all three roles, the benefit of the additional complexity consequent to all three roles' inclusion lies merely in the ability to perform policy derivation, as discussed in Section 7.3.

## 7.2 Tracing Examples

The change classification presented in Section 5.1.1 discussed different types of change a DAS specification may be subject to. One of the benefits of ReAssuRE models is their enhanced tracing support, easing the process of model evolution. This section demonstrates how ReAssuRE models' enhanced tracing support was used to refine the GridStix system design in two scenarios. The first covers a broken assumption stemming from faulty domain understanding, and the second a set of environmental conditions unforeseen at design time.

Assumptions are made extensively when scoping and specifying any system, and the complexity of environments for which DASs prove most useful means that sometimes fundamental assumptions made during early-phase RE can later be found to be (or proved to be) false. ReAssuRE models make explicit the assumptions on which component

selection decisions are based, allowing the impact of a faulty assumption to be assessed and for decisions reliant on broken assumptions to be traced and re-made in light of updated information and new understanding. For the GridStix system, it was discovered after deployment that the GridStix system typically issued flood alerts whilst in the  $S_2$  configuration, and that by the time the GridStix system needed to adapt to  $S_3$  because the nodes were in danger, timely and accurate flood predictions were less important than first expected.

The importance of timely and accurate flood predictions in  $S_3$  was an assumption captured explicitly in the Level-One Claim Refinement Model for  $S_3$ , Figure 49 on page 149. This means that the impact of the claim no longer holding could be assessed using ReAssuRE models, thanks to their support for this type of forward tracing.

The claim in Figure 49 under scrutiny is “Prediction Accuracy is Crucial in  $S_3$ ”. The improved understanding that has emerged through monitoring the GridStix system in operation clearly shows that this claim doesn't hold. Thus, a “broken” label may be applied to the claim, and propagated throughout the model as demonstrated in Section 5.3. The resulting Claim Refinement Model is depicted in Figure 51.

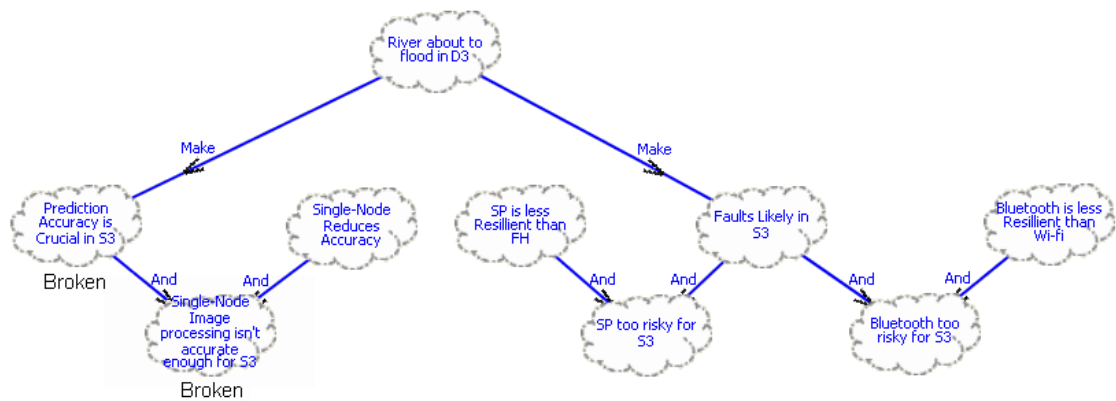


Figure 51: Modified GridStix  $S_3$  Claim Refinement Model: Broken Assumption

After propagation, one-bottom level claim has been broken, which means that the decision(s) reliant on it in the Strategic Rationale model need to be revisited. The Strategic

Rationale model showing the now broken “Single Node Image Processing isn't Accurate Enough for  $S_3$ ” bottom-level claim is depicted in Figure 52.

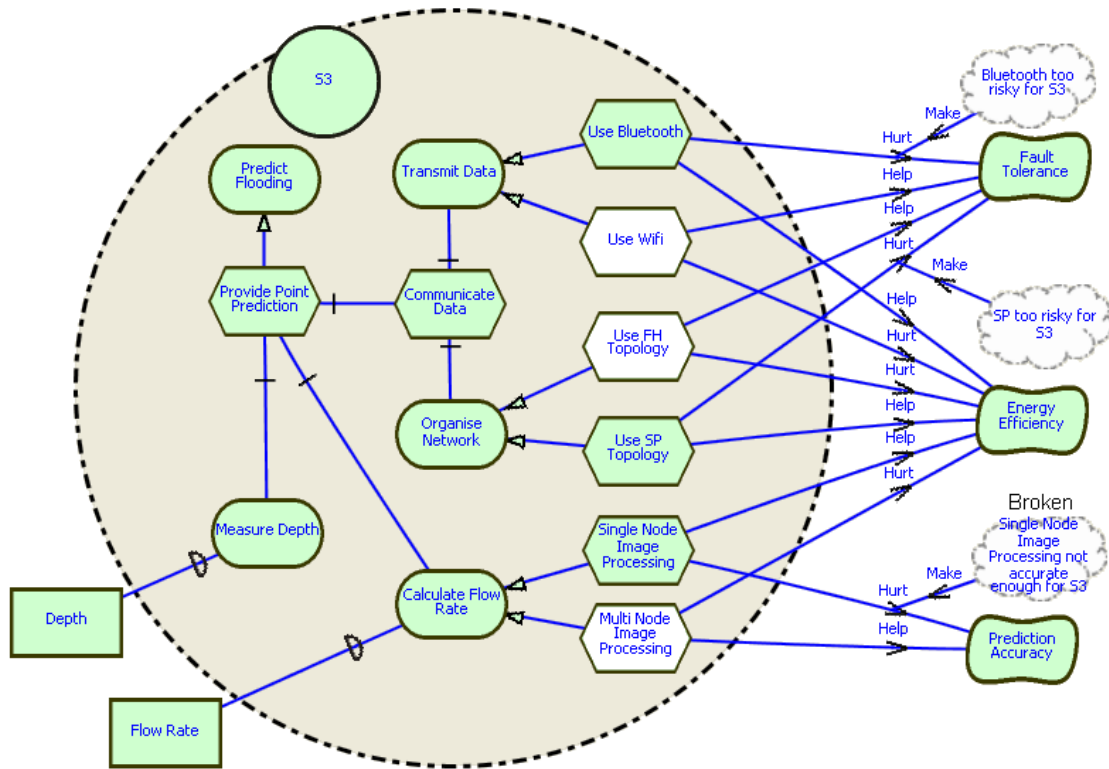


Figure 52: Modified SR Model for GridStix  $S_3$  Target System: Broken Assumption

In Figure 52, the broken bottom-level claim supports the selection of multi node image processing. This decision now needs to be revisited. Given that predictions are now considered less important than survival in the  $S_3$  target system, switching to single node image processing and improving the target system's energy efficiency seems wise. The updated Level-One Strategic Rationale and Claim Refinement Models featuring this changed decision outcome and new rationale are depicted in Figure 53 and Figure 54.

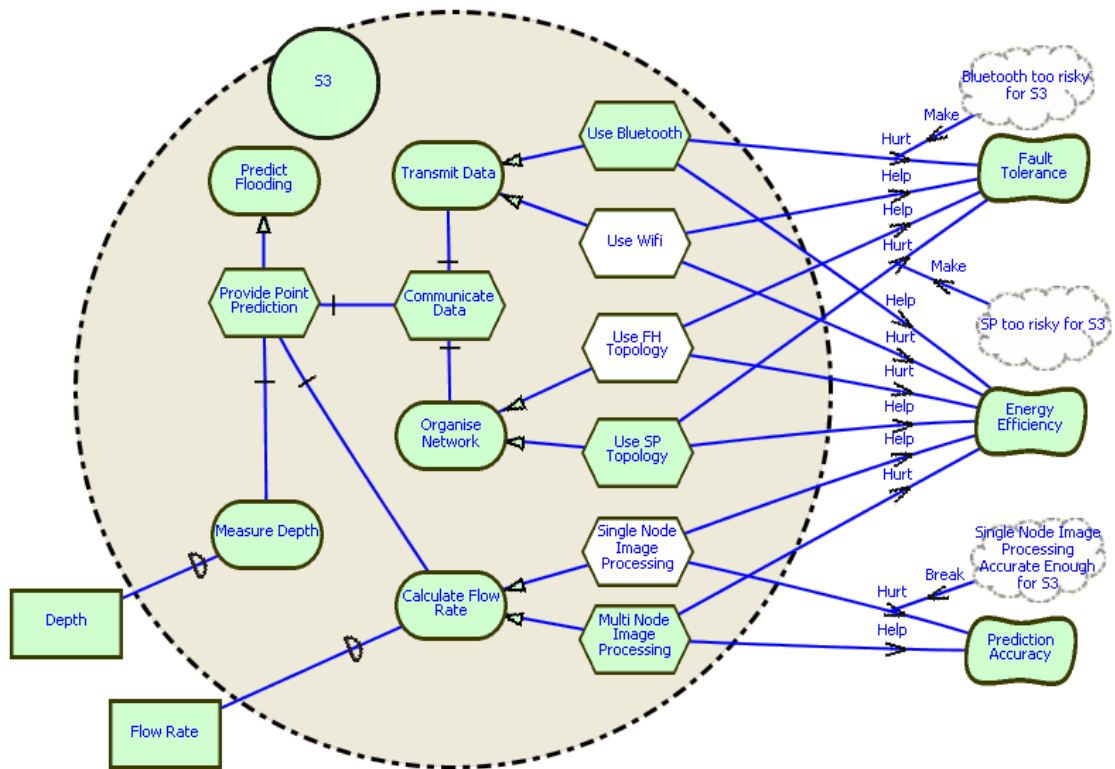


Figure 53: Modified SR Model For GridStix  $S_3$  Target System: New Rationale

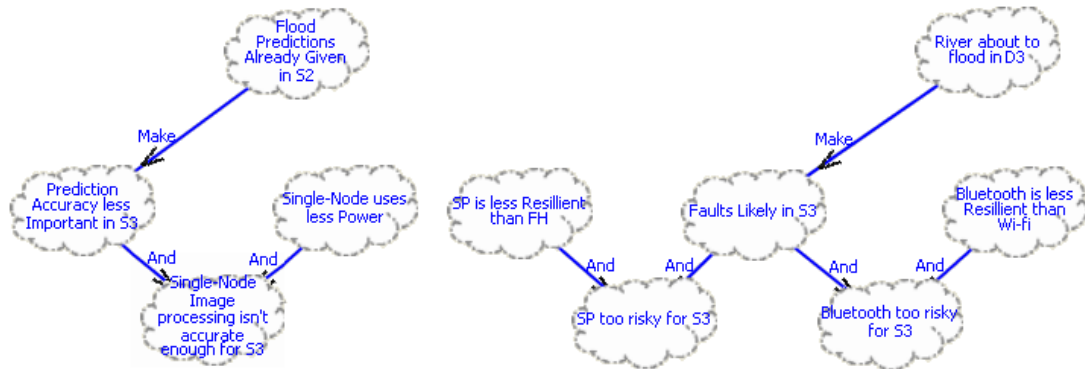


Figure 54: Modified Claim Refinement Model for GridStix  $S_3$ : New Rationale

As discussed in Section 5.1.1, an environmental change to a DAS's specification could take the place of a new environmental constraint, an adjustment to domain boundaries, or

an entire domain being added, removed, or merged with another. This could be driven by improvements in understanding of the environment throughout the DAS's life cycle, or by the environment itself changing over time. For the GridStix system, an unforeseen set of environmental conditions was discovered after deployment, occurring when the river is occluded downstream from the GridStix system. When the river is either completely blocked or the water's flow restricted, either due to a man-made obstacle or debris, conditions upstream deteriorate, with the river depth rising. The GridStix system misidentified this depth increase as a developing flood, leading to a false alarm. However, a blockage downstream from the GridStix nodes reduces the measured flow rate, which the DAS can use to distinguish the  $D_4$  (blockage) domain from  $D_3$  (flood).

Although a river blockage does not necessarily pose a danger to local residents, and should not trigger a flood alert, there is a significant risk of the GridStix nodes becoming submerged or damaged as the river's depth increases, causing them to fail. Furthermore, should the blockage be removed (or breached) suddenly, the river's behaviour could be very difficult to predict without data sourced downstream of the blockage. Therefore, it would be desirable to report the condition to the Environment Agency, the system stakeholder who already receive flood warnings. Out of system scope, the Environment Agency would seek to remove occluding objects and restore the river's natural flow. It could be argued that the emergence of the blockage domain, and the requirement to report its encounter are two separate changes; one environmental and one stakeholder requirements change. However, requiring the DAS to report that it has adopted a particular target system configuration does not add an additional goal to the Level-One models of an individual target system, it adds an additional goal to the Level Two model of the transitions *to* the new target system.

As discussed in Section 5.1.2, the introduction of a new domain requires a new Level-One Strategic Rationale and Claim Refinement Model to be created. Where the target system's goals and available components are the same as another's, the existing Strategic Rationale model may be copied, minus the claims and with the appropriate change to the actor's name to make the model for the new target system. From there, component

selection decisions are taken for the new target system, the outcome of which are recorded on the Strategic Rationale model with claims. The tracing need supported by this process is the ability to identify the component selection decisions that are taken per target system, and ReAssuRE Level-One models' recording of both rejected and selected decision alternatives means that these decisions are easily identified thanks to the presence of several tasks to potentially satisfy a goal via a means-end link on the model. The completed Level-One Strategic Rationale model for the  $S_4$  target system is depicted in Figure 55 below.

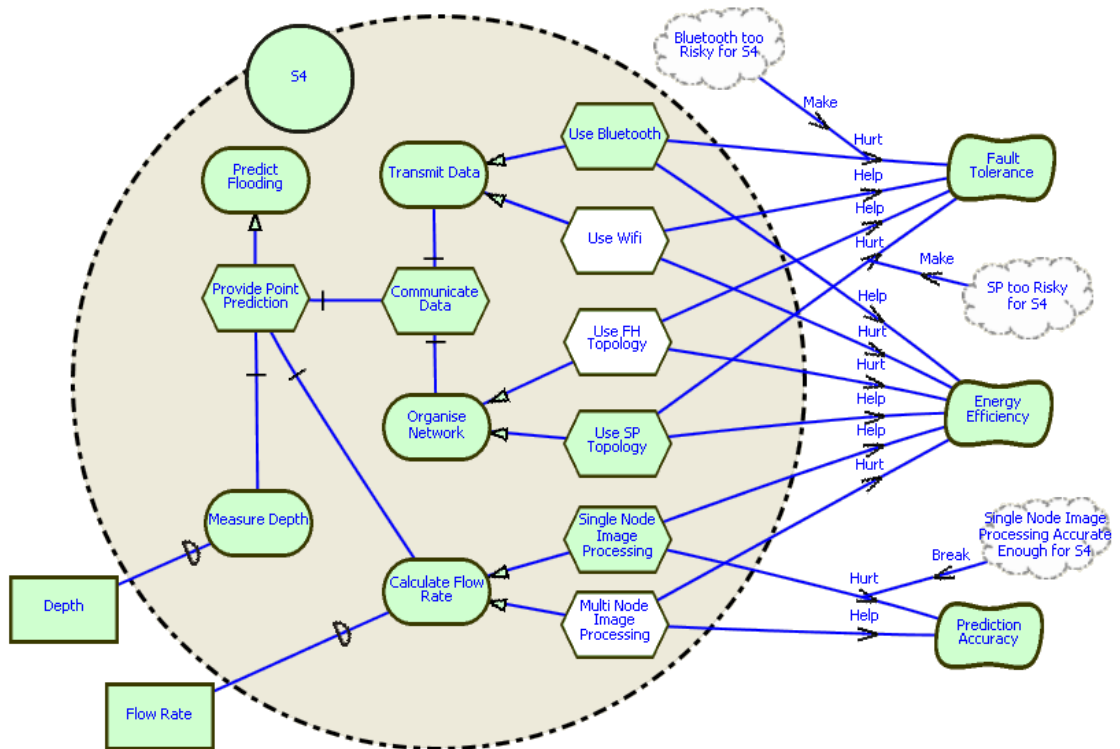


Figure 55: Level 1 SR Model for new GridStix  $S_4$  Target System

As Figure 55 shows, the  $S_4$  target system has been specified as using Wi-Fi for inter-node communication, with the nodes themselves organised using a fewest hop topology. Single node image processing has been specified to derive flow rate measurements from the on-board digital camera. These three specification decision are each justified by a claim. The decisions governing the use of Wi-Fi and fewest hop topology essentially share



the same as their equivalents in the  $S_3$  target system. The final decision, however, is underpinned by a “Single Node Image Processing is Accurate Enough for  $S_4$ ” claim, which eludes to the full rationale which is that, given that local residents are not in danger from a flood, less accurate predictions can be used to save power. The Level-One Claim Refinement Model for the  $S_4$  domain is depicted in Figure 56.

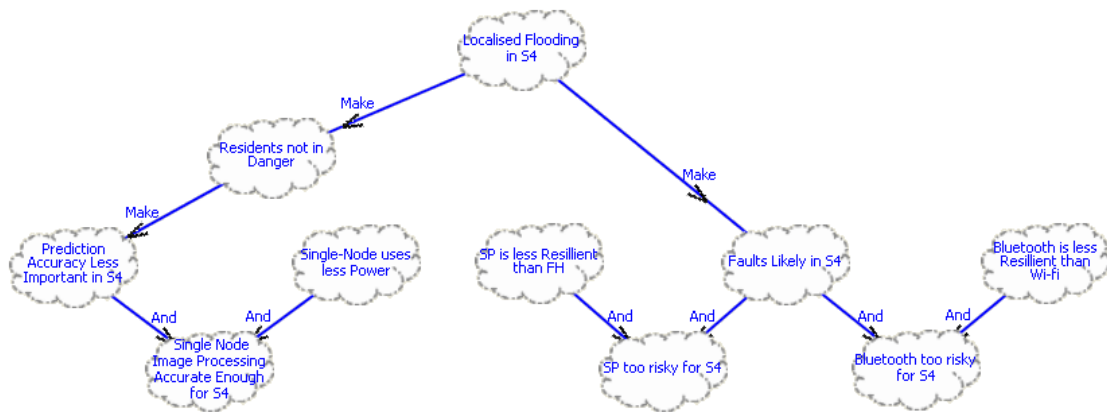


Figure 56: Claim Refinement Model for new GridStix  $S_4$  Target System

Given the rationale behind the decision to use Wi-Fi over Bluetooth, and that to use a fewest hop rather than a shortest path network topology is essentially the same as for the  $S_3$  target system, it is no surprise that the right half of Figure 56 is likewise similar to Figure 49: the claim refinement model for  $S_3$ . The left hand side of the model, uses a claim to assert that although the localised flooding encountered in  $S_4$  presents a danger to the GridStix nodes, no such danger is presented to local residents, and thus prediction accuracy is less important. This, combined with the fact that the GridStix system can save power by using single-node image processing means that the  $S_4$  target system is specified using this, rather than the more accurate but less energy efficient multi-node image processing.

The Level-Two models associated with the  $S_4$  target system are somewhat more interesting than those encountered previously: the  $S_4$  target system can only ever be transitioned *to* from the  $S_1$  target system, but may be transitioned *from* to either the  $S_1$  or

$S_2$  target system, depending on how quickly the blockage is cleared and the level of recent rainfall. The GridStix system may not transition from  $S_2$  to  $S_4$  because the rise in depth would be interpreted as a flood meaning that the  $S_3$  target system was adopted, and the GridStix system may not transition from  $S_3$  to  $S_4$  given that the river has already burst its banks, and any blockage between the banks will simply worsen the flood. Thus, when introducing the  $S_4$  target system, Level-Two models need to be created for the  $S_1$ - $S_4$ ,  $S_4$ - $S_1$  and  $S_4$ - $S_2$  transitions. As mentioned at the start of the blockage tracing example, the stakeholder requirements change to report the detection of a blockage (or rather, that the DAS has adopted the  $S_4$  target system configuration) affects the Level-Two models of the transitions to the  $S_4$  target system, with model covering the  $S_1$  to  $S_4$  transition depicted in Figure 57.

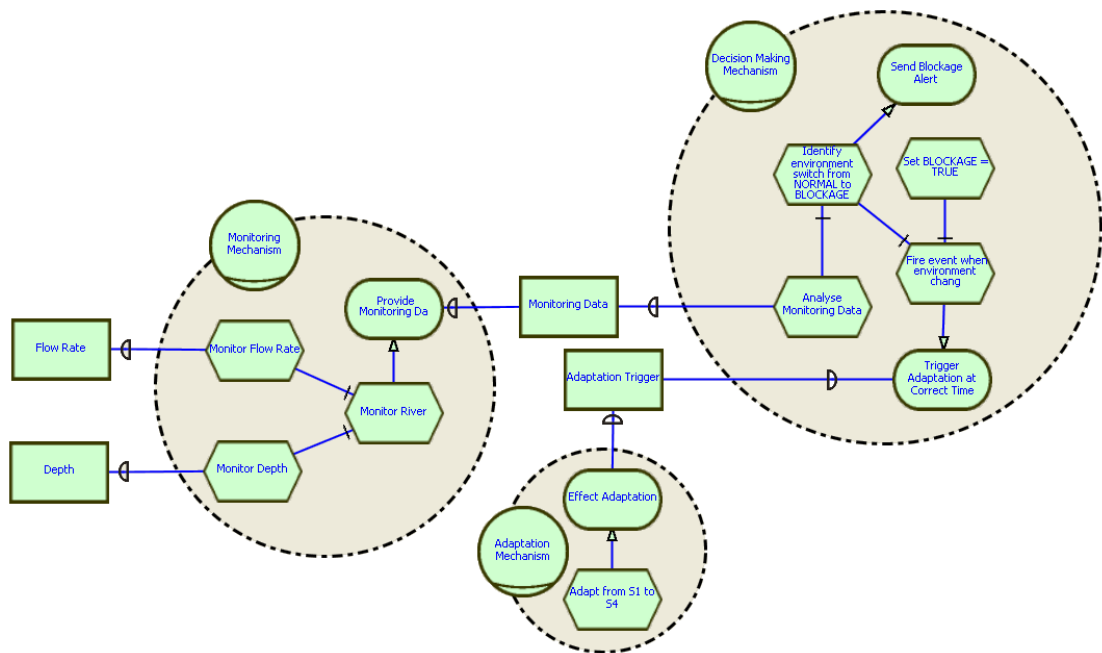


Figure 57: Level Two Model for GridStix  $S_1$  to  $S_4$  Transition

The decision as to which of the three adaptive infrastructure roles depicted in Figure 57 would be assigned responsibility for reporting the blockage was marginal. The monitoring mechanism could have been re-designed to report a low flow rate coupled

with a high depth. Likewise, the adaptation mechanism could have been re-designed to alert stakeholders when adapting from  $S_1$  to  $S_4$ . However, it was decided that, given the decision making mechanism is already required to identify the blockage condition to fire an event triggering adaptation, it should bear responsibility for reporting blockages.

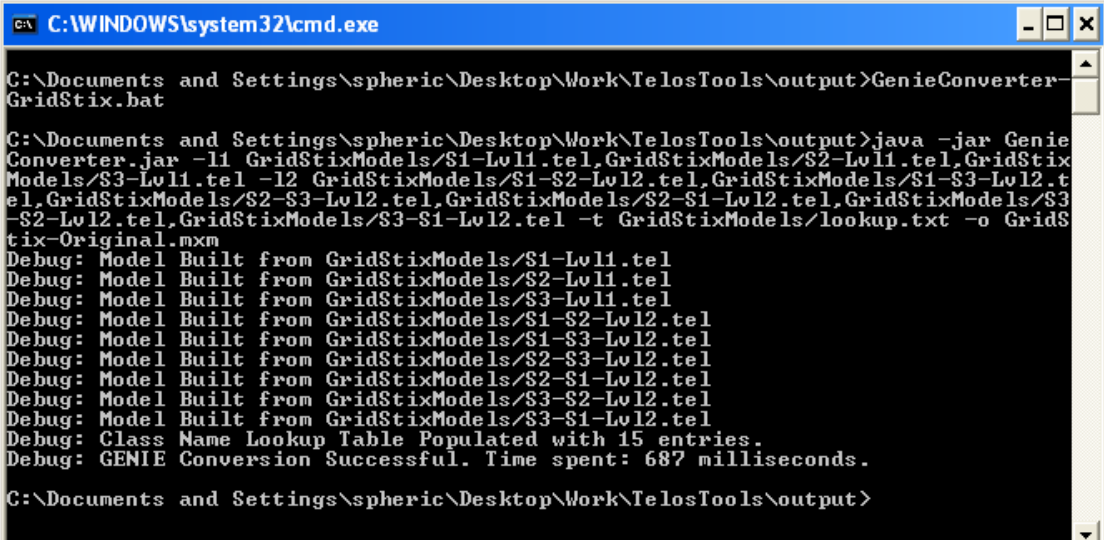
The need to evolve Level-Two models in response to a stakeholder requirements change is a good example of the variable scope of such changes. Some user requirements changes will add a new goal to one or more Level-One Strategic Rationale models. Some may remove a decision alternative from one or more models. More extreme examples may necessitate a complete re-design of the system. It is impossible to fully anticipate and cater for the tracing needs of such varied changes, and this thesis makes no claims of ReAssuRE models' improved traceability easing stakeholder requirements changes in general. The requirement to report blockages, however, is a relatively minor addition which the ReAssuRE models' separation of modelling by level (as specified in [10]) allows to be handled efficiently.

To conclude, this section demonstrates the usefulness of ReAssuRE models' improved traceability in a real DAS that is larger and more complex than the adaptive image viewer used to illustrate the modelling method. The likelihood of DAS specifications being subject to change elevates requirements traceability from the sort of issue important only to large-scale, complex systems to a key requirement of a DAS RE process, model-driven or otherwise. The section demonstrates two different types of tracing activity, demonstrating that the general improved traceability of ReAssuRE models supports different types of tracing activity, and thus eases the implementation of different classes of specification change.

Without the additional tracing information present in ReAssuRE models, the modelling effort involved in performing the illustrated changes would be significantly greater. The increased modelling effort would be further exaggerated in cases where knowledge of decision alternatives considered or of the rationale underpinning decisions has been lost, perhaps by personnel leaving the project.

## 7.3 Policy Derivation

Section 5.2 presented a tool which derived adaptive behaviour from ReAssuRE models, outputting to the Genie DSL [107], which allows adaptive behaviour to be examined, reasoned with, and exported to policies for adaptive middleware platforms. The Level-One Strategic Rationale models for the  $S_1$  (Figure 44),  $S_2$  (Figure 45), and  $S_3$  (Figure 46) target systems, along with the complete set of Level-Two models (for example, Figure 50) were used to derive Genie DSL representation of the GridStix system's adaptive behaviour. Figure 58 shows the tool whilst generating the Genie model.



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>GenieConverter-GridStix.bat

C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>java -jar GenieConverter.jar -l1 GridStixModels/S1-Lvl1.tel,GridStixModels/S2-Lvl1.tel,GridStixModels/S3-Lvl1.tel -l2 GridStixModels/S1-S2-Lvl2.tel,GridStixModels/S1-S3-Lvl2.tel,GridStixModels/S2-S3-Lvl2.tel,GridStixModels/S2-S1-Lvl2.tel,GridStixModels/S3-S1-Lvl2.tel -t GridStixModels/lookup.txt -o GridStix-Original.nxm
Debug: Model Built from GridStixModels/S1-Lvl1.tel
Debug: Model Built from GridStixModels/S2-Lvl1.tel
Debug: Model Built from GridStixModels/S3-Lvl1.tel
Debug: Model Built from GridStixModels/S1-S2-Lvl2.tel
Debug: Model Built from GridStixModels/S1-S3-Lvl2.tel
Debug: Model Built from GridStixModels/S2-S3-Lvl2.tel
Debug: Model Built from GridStixModels/S2-S1-Lvl2.tel
Debug: Model Built from GridStixModels/S3-S2-Lvl2.tel
Debug: Model Built from GridStixModels/S3-S1-Lvl2.tel
Debug: Class Name Lookup Table Populated with 15 entries.
Debug: GENIE Conversion Successful. Time spent: 687 milliseconds.

C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>
```

Figure 58: Screen-Shot Genie DSL Model Derivation Tool

The resultant Genie model is depicted in Figure 59.

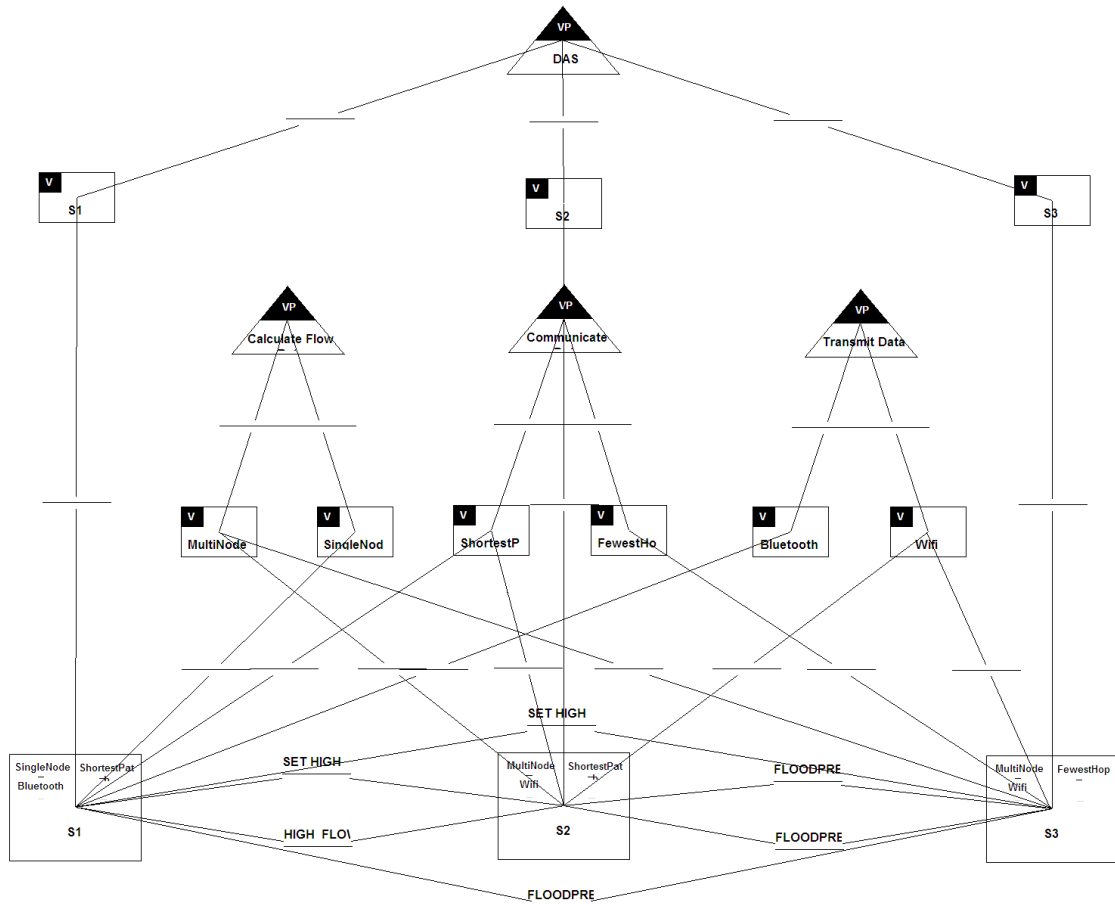


Figure 59: GENIE Model of GridStix DAS Derived from ReAssuRE Models

The Genie model in Figure 59, visualised in the MetaEdit+ modelling tool [108], shows a brief overview of the variation points in the GridStix system, the configuration adopted in each target system and the triggers causing the DAS to switch between target systems.

The GridStix system uses the GridKit middleware [39], upon which it relies for its adaptive capabilities. The automated policy generation tool introduced in chapter 5.2 can derive GridKit policies from a collection of ReAssuRE Level-One strategic rationale models, and Level Two models, and can be used to do so here. By verifying the generated policies against those created by hand for the GridStix system, it is possible to confirm that the policy generation method automated by the tool yields policies are correct.

The policy generation tool was run using the ReAssuRE models presented in Section 7.1, as used for the Genie conversion tool. Figure 60 shows the tool generating the policies, and Figure 61 shows a small excerpt from the generated adaptation policy. The full XML policy is included as Appendix B – GridStix Adaptation Policy.

```

C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>PolicyConverter
-GridStix.bat
C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>java -jar Polic
yConverter.jar -l1 GridStixModels/S1-Lvl1.tel,GridStixModels/S2-Lvl1.tel,GridSti
xModels/S3-Lvl1.tel -l2 GridStixModels/S1-S2-Lvl2.tel,GridStixModels/S1-S3-Lvl2.
tel,GridStixModels/S2-S3-Lvl2.tel,GridStixModels/S2-S1-Lvl2.tel,GridStixModels/S
3-S2-Lvl2.tel,GridStixModels/S3-S1-Lvl2.tel -t GridStixModels/lookup.txt -f Span
ningTree -o GridStix-Original.xml
Debug: Model Built from GridStixModels/S1-Lvl1.tel
Debug: Model Built from GridStixModels/S2-Lvl1.tel
Debug: Model Built from GridStixModels/S3-Lvl1.tel
Debug: Model Built from GridStixModels/S1-S2-Lvl2.tel
Debug: Model Built from GridStixModels/S1-S3-Lvl2.tel
Debug: Model Built from GridStixModels/S2-S3-Lvl2.tel
Debug: Model Built from GridStixModels/S2-S1-Lvl2.tel
Debug: Model Built from GridStixModels/S3-S2-Lvl2.tel
Debug: Model Built from GridStixModels/S3-S1-Lvl2.tel
Debug: Class Name Lookup Table Populated with 15 entries.
Debug: Policy Conversion Successful. Time spent: 563 milliseconds.
C:\Documents and Settings\spheric\Desktop\Work\TelosTools\output>

```

Figure 60: Tool Generating Adaptation Policies from ReAssuRE Models

```

<ReconfigurationRule>
  <FrameWork>SpanningTree</FrameWork>
  <Events><Event>
    <Type>FLOODPREDICTED</Type>
    <Value>TRUE</Value>
  </Event></Events>
  <Reconfiguration>
    <FileType>Java</FileType>
    <Name>Reconfigurations.FewestHop</Name>
  </Reconfiguration>
</ReconfigurationRule>

```

Figure 61: Snippet of Generated XML Adaptation Policy

This snippet of the adaptation policy contains a single reconfiguration rule, which specifies that when transitioning to the  $S_3$  target system that a fewest hop network topology should be adopted. The first piece of information in the reconfiguration rule is the configuration framework to which the reconfiguration applies. For the GridStix system, all reconfigurations use the “SpanningTree” framework. The Event triggering the reconfiguration is the setting of the “FLOODPREDICTED” flag to “TRUE”, which the GridKit middleware implements as an event. The actual code executed to perform the reconfiguration resides in the “FewestHop” class in the “Reconfigurations” package, the execution of which is dictated by the policy above.

Upon Inspection of the (full) generated adaptation policy, it is possible to check that the reconfigurations specified will yield the target system configurations in the ReAssuRE Level-One Strategic Rationale models. Furthermore, it is also possible to check the generated adaptation policy against that created by hand for the GridStix system. Both checks indicate that the generated policy is correct. The behaviour of the GridStix system under set environmental conditions, using different adaptation policies can be assessed using simulation. The next section discusses the simulator in more detail.

## 7.4 Policy Simulation

The GridStix system is deployed in a remote, hostile, environment which makes performing maintenance difficult. Although the GridStix system can report monitoring data, there is little possibility of performing tuning based upon it with any degree of frequency or regularity. Alterations to the GridStix system's adaptive behaviour may have far-reaching consequences in terms of the system's ability to survive its hostile environment, or its success in performing its key goal: predicting flooding. Thus, it is beneficial to simulate candidate adaptation policies before deployment, allowing developers to assess the performance of the GridStix system under different configurations, the long-term consequences of a change of policies, to uncover any emergent behaviour and to allow challenging environmental conditions previously

encountered to be studied to better understand how the GridStix system should self-optimize if they are encountered again.

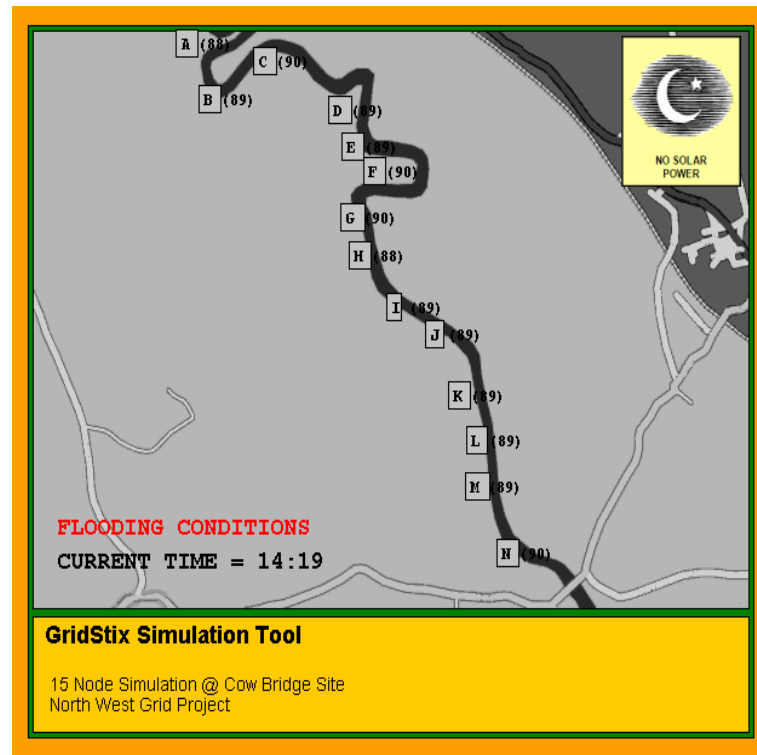
When creating the GridStix system, developers created a basic simulator capable of calculating the energy consumed by nodes when communicating different data payloads using different network topologies, or by using different radio transmission hardware. This simulator was not in itself sufficient to assess the impact of different adaptation policies, the need for which was identified when performing maintenance. This thesis presents an extended version of the simulator, which is able to load adaptation policies and simulate the behaviour of the GridStix system using the policies when exposed to various environmental conditions. The behaviour of the environment may also be scripted, which allows previously encountered environmental conditions to be used to test candidate adaptation policies. The following paragraphs discuss the simulator's capabilities individually, in greater detail.

For the thesis's purposes, the most important simulator feature is policy simulation. This allows adaptation policies to be loaded and interpreted, and configures the simulated nodes as the policy specifies. Internally, the simulator raises adaptation triggers in much the same way as the actual system, with events raised when the simulated nodes detect either a high flow rate or an imminent local flood. Note, that the flood trigger refers to the site at which the GridStix nodes are situated, not the potential flooding downstream that the GridStix system seeks to predict. Thus, adaptation policies may dictate target system configurations for the three identified domains, but not alter the domain partitioning. To simulate the GridStix system's behaviour if the domain partitioning is adjusted, the simulator will need to be modified to raise the correct adaptation triggers for the new partitions.

Another key feature of the simulator is the ability to simulate the nodes' energy usage when communicating with each other, and performing flow rate analysis. This feature was present in the original version of the simulator, but is the feature that allows the impact of configuration changes to be analysed. The energy-usage simulation has been extended to force nodes that exhaust their battery power to fail, which of course affects the operation



of the GridStix system and the performance of the other nodes, thanks to the inherent change in network topology. Figure 62 shows the simulator's visualisation of the GridStix nodes, their remaining power, and the currently active network topology.



*Figure 62: GridStix Simulator Display*

From an environmental perspective, a key but often overlooked feature is the simulation of differing light levels. Naturally, the performance of the GridStix nodes' solar panels is dependent on the amount of sunlight that falls upon them. At night, virtually no power is supplied by the solar panels, during the daytime the amount of energy generated is dependent on the amount of cloud cover. The simulator uses three broad light levels to allow night-time, cloudy and sunny conditions to affect the amount of energy replenishing the nodes' batteries. These light levels form part of the scripted environmental conditions that different target system configurations can be subjected to.

Underpinning the simulator's ability to simulate the GridStix system's adaptive capability is its ability to specify the behaviour of the river. The depth and flow rate of the river can be controlled as part of the environmental script, the simulated nodes monitor them, which in turn prompts the DAS to switch between target systems.

One final feature of the simulator, which is particularly useful for constructing and simulating what-if? scenarios, is the ability to force nodes to fail due to external (or unknown) causes. The deployed GridStix nodes operate in an environment that is not only volatile in terms of its behaviour, but also dangerous to the nodes' hardware. Nodes can fail due to water-borne debris when the river floods, due to the temperature extremes to which they are subjected, or even due to bovine intervention. Thus, the ability to investigate the behaviour of the GridStix system after losing one or more nodes, and to assess whether different target system designs are better able to tolerate node loss, is essential.

To conclude, this section presents an enhanced GridStix simulator that allows different adaptation policies to be trialled under scripted environmental conditions. The simulator is useful in assessing the impact of changes to the GridStix system's adaptive behaviour, and in refining target system designs in light of the challenging environmental conditions the actual system has encountered. The simulator serves also to allow the adaptation policies derived as part of the requirements validation process described in the following section to themselves be validated, and likewise for modified target system designs that a m-DAS may construct under certain scenarios (discussed in Section 7.6).

## 7.5 Requirements Validation

As discussed in Section 5.3, the ReAssuRE models' explicit capturing of assumptions using claims supports requirements validation activity, by exploring what-if? scenarios, in which the behaviour of the DAS is explored when assumptions upon which individual target system designs are based are broken. This type of validation activity has two

benefits: firstly, an assumption in which a low degree of confidence is held is essentially a marker for uncertainty or incomplete understanding of the environment or the expected DAS behaviour. Thus, identifying less certain claims eases the identification of these patchy areas. Secondly, the behaviour of the DAS in adverse conditions can be explored, which allows any deficiencies to be mitigated by improving the robustness of the requirements specification. Section 5.3 noted that this thesis does not prescribe the nature of validation scenarios constructed, highlighting test cases, simulation and static reasoning as likely candidates to support this validation activity. The GridStix simulator introduced in the previous section serves as a useful tool in exploring the GridStix system's behaviour in customisable conditions. This section constructs identified validation scenarios for the GridStix system as simulator runs.

This section will identify and present the validation scenarios constructed for GridStix's  $S_3$  target system, and will work through the validation scenario pruning method discussed briefly in Section 5.3, demonstrating the reduction in validation workload that can be achieved by targeting only the most likely of scenarios.

Referring back to the Level-One Strategic Rationale model for the  $S_3$  target system (Figure 46 on page 146), the  $S_3$  target system is specified as using Wi-Fi for radio communications, a fewest-hop network topology and multi-node image processing to calculate the river's flow rate. The rationale behind the decisions is recorded in the associated Claim Refinement Model, which was depicted in Figure 49 on page 149.

Although it would perhaps be most beneficial to construct validation scenarios covering every combination of supporting claims holding and being invalidated, this thesis advocates constructing validation scenarios only for combinations of bottom-level claims. This is a pragmatic decision to keep validation workloads manageable. A bottom-level claim is usually a composite claim, derived from several underlying claims, and when a bottom-level claim is invalidated it is usually as a result of one or more supporting claims having previously been invalidated also. The effects of broken assumptions thus form clusters, with all the potential combinations of invalidated supporting claims that result in a bottom-level claim (or combination of bottom-level claims) becoming invalidated

having substantially similar effects on the DAS. As such, the focus of the validation work is the three bottom-level claims in Figure 49.

Section 5.3 stated that the number of required validation scenarios ( $T$ ) could be calculated from the number of bottom-level claims ( $n$ ) using the formula:  $T=2^n-1$ , with one being subtracted to exclude the scenario in which all claims hold: the originally envisaged domain. Thus, for the  $S_3$  target system, there are seven possible validation scenarios, as listed in Table 5 below.

Scenario	“Single-Node Not Accurate Enough for $S_3$ ” Claim	“SP too Risky for $S_3$ ” Claim	“Bluetooth too Risky for $S_3$ ” Claim
1	Holds	Holds	Invalidated
2	Holds	Invalidated	Holds
3	Invalidated	Holds	Holds
4	Holds	Invalidated	Invalidated
5	Invalidated	Holds	Invalidated
6	Invalidated	Invalidated	Holds
7	Invalidated	Invalidated	Invalidated

*Table 5: Claim Combinations for GridStix  $S_3$  Validation Scenarios*

Although seven validation scenarios do not represent an insurmountable validation burden, there are also those scenarios for the other target systems to consider. Table 6 below shows the number of validation scenarios required for each of the GridStix target systems, and in total.

Target System	Required Validation Scenarios
$S_1$	31
$S_2$	7
$S_3$	7
<b>Total</b>	<b>45</b>

Table 6: GridStix Validation Scenarios by Target System

In Table 6, the number of validation scenarios needed for the  $S_1$  target system stands out as being significantly greater than those for other target systems. This large number of validation scenarios is because of the style of claim notation used in the Level-One strategic rationale model for  $S_1$ , which used several bottom-level claims to convey a more complex rationale.  $S_2$  and  $S_3$  use a single inclusive claim to support each individual decision, which reduces the number of validation scenarios significantly. Thus, there is a tension between the desire to maximise the precision of the recorded rationale (which often requires more claims) and the desire to minimise the number of validation scenarios.

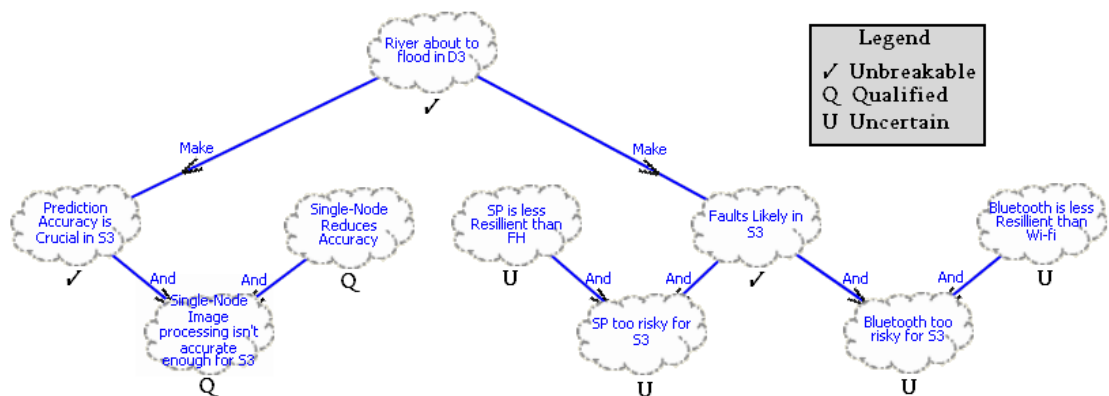


Figure 63: GridStix  $S_3$  Claim Refinement Model with Claim Classification

Clearly, forty five validation scenarios for what is a relatively simple DAS is undesirable, and more complex DASs could require significantly more. Thus, the

validation scenario pruning method discussed briefly in Section 5.3 shall be used. The validation scenario pruning method requires claims on the claim refinement model to be classified according to the level of confidence held in each. Figure 63 shows the claim refinement model for the  $S_3$  target system (as in Figure 49) annotated with the claim confidence labels assigned.

In Figure 63 there are no unbreakable bottom-level claims, one qualified and two uncertain. The most important classification decisions occur on the middle level of Figure 63, with “Single-Node [Image Processing] Reduces Accuracy” being classed as qualified, and the “SP is less Resilient than FH” and “Bluetooth is less Resilient than Wi-Fi” claims being classed as uncertain.

Single-node image processing reduces prediction accuracy due to the greater time needed between measurements to allow the processing to be completed on a single node reduces both timeliness and the amount of data available to the prediction algorithm, and as such the “Single-Node Reduces Accuracy” claim was considered qualified. The two uncertain claims were classified as such because the analyst was unsure as to whether the theoretically more resilient option in each case would prove so demonstrably in the field.

With no unbreakable claims, one qualified and two uncertain, removing unbreakable bottom-level claims from consideration will yield be no reduction in the required validation scenarios. If qualified claims are also removed from consideration, three validation scenarios shall be needed. Table 7 below shows the results of the same working on each of the GridStix target systems, and in total. The annotated claim refinement models for  $S_1$  and  $S_2$  are included as Appendix C – Annotated GridStix Claim Refinement Models.

Target System	All Claims	Excluding Unbreakable Claims	Excluding Qualified & Unbreakable Claims
S <sub>1</sub>	31	7	3
S <sub>2</sub>	7	3	3
S <sub>3</sub>	7	7	3
<b>Total</b>	45	17	9

*Table 7: GridStix Validation Scenario Reduction using Claim Classification*

Table 7 shows that for the GridStix system as a whole, removing unbreakable claims from consideration when generating validation scenarios will reduce the number of scenarios needed from 45 to 17. If qualified claims are also excluded, the number of required scenarios will drop further to 9. Returning to the S<sub>3</sub> target system, the three remaining validation scenarios centre around the claims that Bluetooth communication and a shortest-path network topology present too much of a risk in terms of fault tolerance to be used in the S<sub>3</sub> target system, with the three key uncertain validation scenarios covering one, other or both of these claims no longer holding. This section discusses the two scenarios stemming from a single claim no longer holding in turn.

As discussed at the start of the section, this thesis uses simulation as the means to construct validation scenarios. To simulate the expected environmental conditions for the S<sub>3</sub> target system, a combination of meteorological and logged data from the actual system in operation was used to construct a simulator environment script of a fairly typical flood event from start to finish.

The simulated conditions start from a period of relatively good weather, which means that the GridStix nodes start the simulation with close to full batteries. Then, a period of heavy rainfall occurs, lasting approximately 72 hours. During this time, the poor lighting conditions mean that the GridStix nodes' solar panels perform poorly. During this time, the river's flow rate steadily increases, as rainfall upstream flows past the site. After the 72 hours elapse, the conditions do not improve, but the river depth starts to rise, which

causes the GridStix system to adopt the  $S_3$  target system configuration. After approximately 36 hours of the river's depth slowly increasing, GridStix nodes begin to become submerged, and two nodes soon fail. (In the real-life scenario reflected by these conditions, one failed due to flawed waterproofing and the second due to debris damage). The weather eventually improves, and the river's depth peaks before all the nodes become submerged, and as the flood subsides the GridStix system adopts the  $S_2$  target system configuration.

The two validation scenarios consist of a simulator run using the described environmental conditions, the first run simulating poorer than expected Wi-Fi performance by reducing the achievable communications range to little above that achieved by Bluetooth (essentially invalidating the “Bluetooth is Less Resilient than Wi-Fi” claim in Figure 63).

This first simulator run indicated that although the GridStix system was able to survive long enough to predict a flood and adapt to  $S_3$ , the node failures occurring soon after the flood event begins caused two nodes (G and J in Figure 62 on page 166) to drain their battery power significantly faster than the other nodes, thanks to the increased communications traffic flowing between them after nearby nodes had failed. Although the simulated flood event subsides before the nodes exhaust their battery power, the results indicate that longer floods would see these nodes fail in similar circumstances. Given that the invalidated bottom-level claim under study supports selecting Wi-Fi over Bluetooth, a second simulator run was conducted with Bluetooth used for communications instead. These results indicated that the GridStix system would survive for two to three days longer in these conditions using Bluetooth. However, if node G or J were to fail during the flood event, the network would become fragmented. Thus, the results of the validation scenario are that the consequences of the “Bluetooth too Risky for  $S_3$ ” claim not holding are not catastrophic, which means that additional effort should not be expended in verifying the supporting claims.

The second simulator run aimed to assess the performance of the GridStix system should a fewest hop network topology prove no more resilient than shortest path. This



was achieved by changing the  $S_3$  network topology to shortest path (so in this run, the fewest hop topology is *exactly* as resilient as shortest path) and applying an additional power drain so that the nodes were consuming power as if communicating by fewest hop, whilst receiving only the fault tolerance level of shortest path. This simulation is imperfect because the distribution of power consumption between nodes would differ in a fewest-hop topology. The simulation results included several node failures due to battery exhaustion, eventually fragmenting the network. As such, the validation scenario indicates that the consequences of the “SP too risky for  $S_3$ ” claim are potentially serious, and that some validation activity should be undertaken to ensure that the supporting claims are sound.

This second scenario was challenging to construct and visibly imperfect, it should also be noted that a simulator run designed to validate the claim would actually have been simpler in this instance. As such, this thesis recommends that care be taken when constructing validation scenarios; in cases where effort can better be expended in validating the assumptions codified by claims it is better to do this than expend effort exploring the consequences of the claim not holding.

The outcomes of the two validation scenarios are relatively clear: the consequences of the “Bluetooth too Risky for  $S_3$ ” claim being invalidated are not severe, whereas the consequences of the “SP too Risky for  $S_3$ ” claim are. Thus, validation effort should be expended on validating the latter by validating its supporting claims.

To conclude, this section demonstrates how validation scenarios may be identified and constructed for a target system by analysis of Level-One claim refinement models. It has highlighted the explosive increase in validation burden as the number of bottom-level claims in ReAssuRE models increases, and identified a tension between the desire to maximise the precision of recorded rationale and the desire to minimise validation workload. In mitigation, the section demonstrates the significant reduction in the number of validation scenarios that can be achieved by excluding claims in which a greater degree of confidence is held, targeting validation effort at the more uncertain claims.

The systematic identification of areas of uncertainty within a DAS specification was considered useful by GridStix's developers, particularly in conjunction with the previously discussed possibility of uncovering implicit assumptions when first constructing the Claim Refinement Models. The system developers felt, however, that the number of validation scenarios required to cover all combinations of broken and held assumptions was burdensome, and as such the simple method of prioritising key scenarios was welcome.

The section shows also that validation scenarios can be difficult to construct for a given combination of claims holding or otherwise, and that in some cases it may be beneficial to assume the outcome of a validation scenario indicating a need for claim validation when said validation can be performed more easily and effectively.

## 7.6 Model-Driven Adaptation

Chapter 6 discusses the use of ReAssuRE models at run time to guide a DAS's adaptation. This thesis proposes the monitoring of assumptions in a manner analogous to requirements monitoring [30], modifying Level-One Claim Refinement Models to reflect operating conditions when assumptions are found to no longer hold by invalidating the claims representing broken assumptions, and re-deriving adaptation policies from the changed models to yield new adaptive behaviour. This section demonstrates the model modifier discussed in Section 6.3, and shows that, in some circumstances, a m-DAS version of the GridStix system can adapt to and is better equipped to tolerate unforeseen environmental conditions.

In devising a m-DAS using assumption monitoring, the first step to be taken is the identification of assumptions for which monitors can be devised. Returning to the claim refinement model for GridStix's  $S_3$  target system, depicted in Figure 49 on page 149, there are nine claims in total. However, not all of these claims prove monitorable.

Of the claims in Figure 49, just five are directly monitorable, and monitoring one of the five would be needless. The “River about to flood in  $D_3$ ” claim is, of course, already

monitored by the DAS to allow transitioning out from the  $S_3$  target system. As such, the claim is treated as un-monitorable. The four remaining monitorable claims are: “Single-Node [Image Processing] Reduces Accuracy”, “SP is less Resilient than FH”, “Faults Likely in  $S_3$ ” and “Bluetooth is less Resilient than Wi-Fi”.

Monitoring the “Single-Node [Image Processing] Reduces Accuracy” claim would be a matter of maintaining a record of the percentage of predicted floods that occur, or the percentage of actual floods not predicted whilst using multi-node and single-node image processing. In the event of the two percentages converging, or even the single-node prediction accuracy being found higher, the claim could be invalidated.

Monitoring of the “SP is less Resilient than FH” claim would take the form of recording the number of GridStix nodes that need to fail before any remaining functional nodes become separated from the main network, unable to communicate. Given that network partitions impact the ability of the GridStix system to predict floods, the GridStix system is designed to minimise this occurrence. Thus, it is expected that if at all, this assumption would only be invalidated over time. Monitoring of the “Bluetooth is less Resilient than Wi-Fi” claim can be similarly achieved.

Monitoring the “Faults Likely in  $S_3$ ” claim is relatively trivial, and can be achieved by recording the number of node failures in  $S_3$ , and comparing the number with those for other target systems.

This thesis does not prescribe the form of assumption monitors, much as the main body of requirements monitoring work does not prescribe the form of requirement monitors [30] [92]. The potential monitors discussed above are included merely to illustrate the level of detail that needs to be considered when classifying claims as monitorable or otherwise. The model modifier discussed in Section 6.3 adjusts Claim Refinement Models in response to events raised by monitors, propagating a “Broken” label throughout the model. Should a bottom-level claim be invalidated either directly by a monitoring event or indirectly via label propagation, the associated Strategic Rationale model is modified in accordance with a monitoring policy. Figure 64 depicts a portion of

the monitoring policy for the GridStix  $S_3$  target system, covering the monitors and modification for two of the four identified monitored claims.

```

<MonitoringPolicy>
  <MonitorRule>
    <Monitor>
      <Event>gridstixsupport.S3BTLessResWifiEvent</Event>
    </Monitor>
    <Transformation>
      <Node type="claim">
        Bluetooth is less resilient than Wifi
      </Node>
      <Action>INVERT</Action>
    </Transformation>
  </MonitorRule>
  <MonitorRule>
    <Monitor>
      <Event>gridstixsupport.S3NoFaultsEvent</Event>
    </Monitor>
    <Transformation>
      <Node type="claim">
        Faults Likely in S3
      </Node>
      <Action>INVERT</Action>
    </Transformation>
  </MonitorRule>
</MonitoringPolicy>

```

*Figure 64: Snippet of GridStix Monitoring Policy*

Each of the monitor rules in Figure 64 stipulate that on receipt of the appropriate event (the event objects are packaged in the “gridstixsupport” class), a specific claim on the Claim Refinement Model should be invalidated. If the “Broken” label reaches a bottom-level claim as a result of this model modification, the model modifier should invert the

contribution of the bottom-level claim on the Strategic Rationale model. Figure 65 and Figure 66 depict the Level-One Claim Refinement and Strategic Rationale models respectively, after the model modifier has acted on receipt of a “S3BTLessResWifiEvent”, which would be raised by the assumption monitor covering the “Bluetooth is Less Resilient than Wi-Fi” claim.

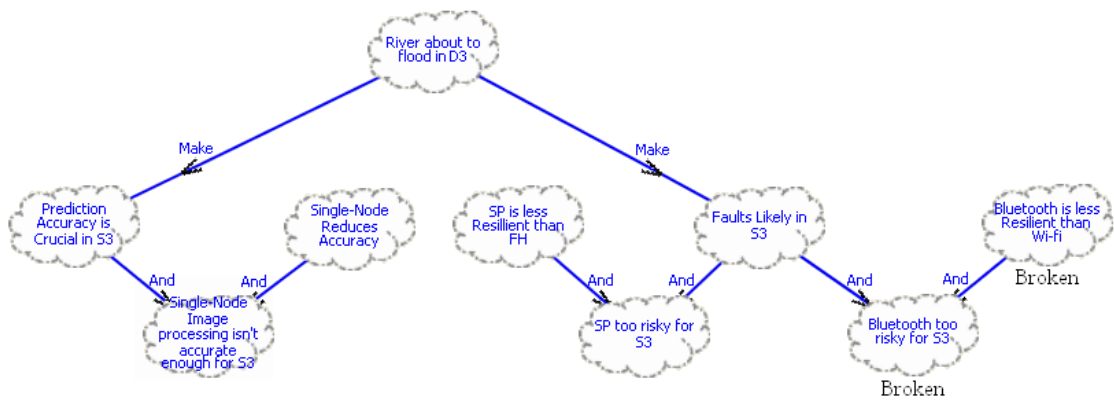


Figure 65: Modified GridStix  $S_3$  Claim Refinement Model

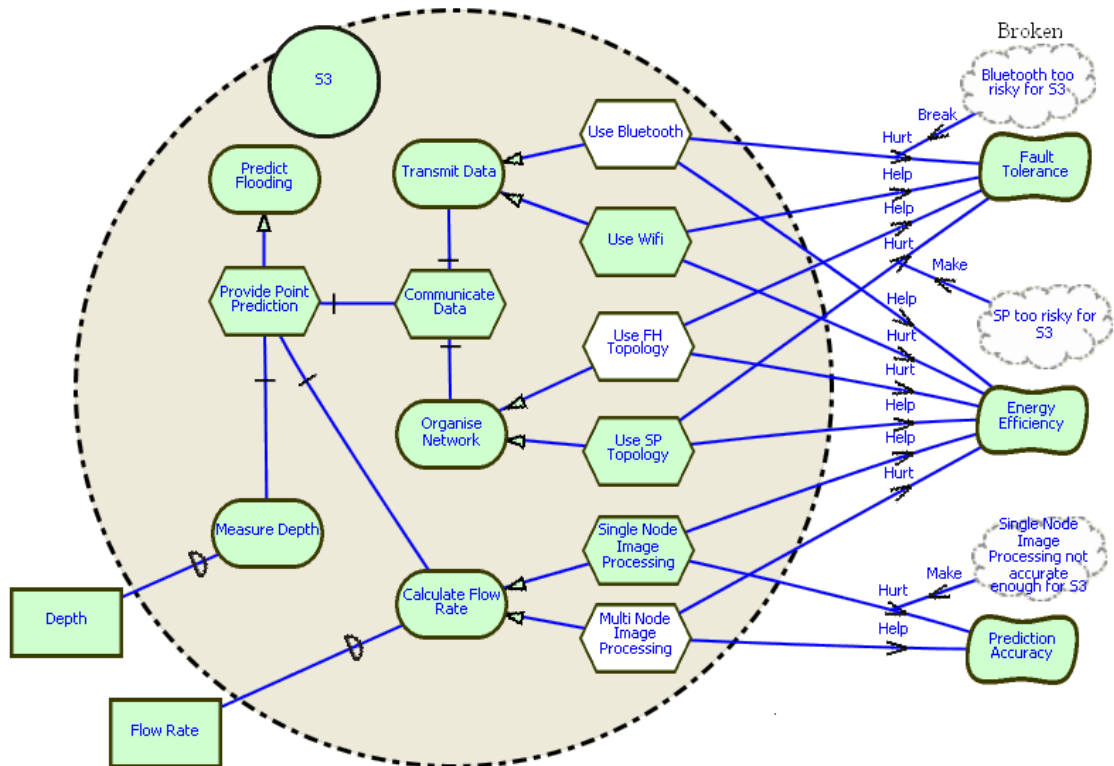


Figure 66: Modified GridStix  $S_3$  Strategic Rationale Model

Figure 65 shows the “Broken” label having been applied to the “Bluetooth is less resilient than Wi-Fi” claim, from where it propagated to the “Bluetooth too risky for  $S_3$ ” bottom-level claim. Figure 66 shows that the now invalidated bottom-level claim's contribution to the strategic rationale model has been inverted: a “Break” contribution as opposed to a “Make” on the original model. The inverted contribution means the claim now supports the selection of Bluetooth instead of Wi-Fi in the original  $S_3$  model. In short, these model modifications would mean that if an assumption monitor determined that under current conditions, Wi-Fi radio communications were proving no more resilient than Bluetooth, the m-DAS would adjust the  $S_3$  target system design temporarily to use Bluetooth, saving power.

Using the model modifier with the monitoring policy depicted in Figure 64, it is possible for a m-DAS to produce the modified models depicted in Figure 65 and Figure 66

autonomously. Using the policy generation method discussed in Section 5.2, the m-DAS may re-derive updated adaptation policies to reflect the new configuration supported by the modified models. Running the new adaptation policies in the simulator scenario discussed in the previous section, in which Wi-Fi significantly under-performs, indicates that the GridStix nodes would survive longer in this configuration under the simulated conditions, due to the lower power usage.

In the second example, Figure 67 and Figure 68 show the  $S_3$  Claim Refinement and Strategic Rationale models, respectively, after the model modifier has acted on receipt of a “S3NoFaultsEvent”, which would be raised by an assumption monitor covering the “Faults Likely In  $S_3$ ” claim.

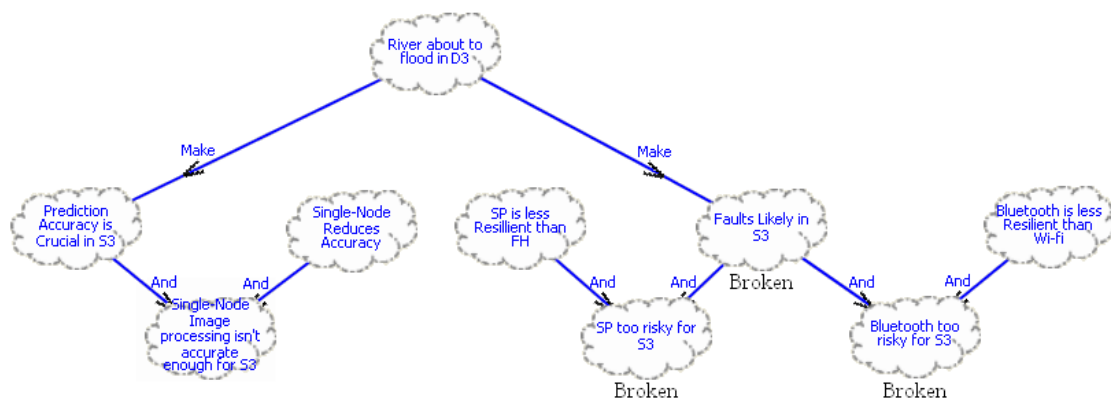


Figure 67: More Extensively Modified GridStix  $S_3$  Claim Refinement Model

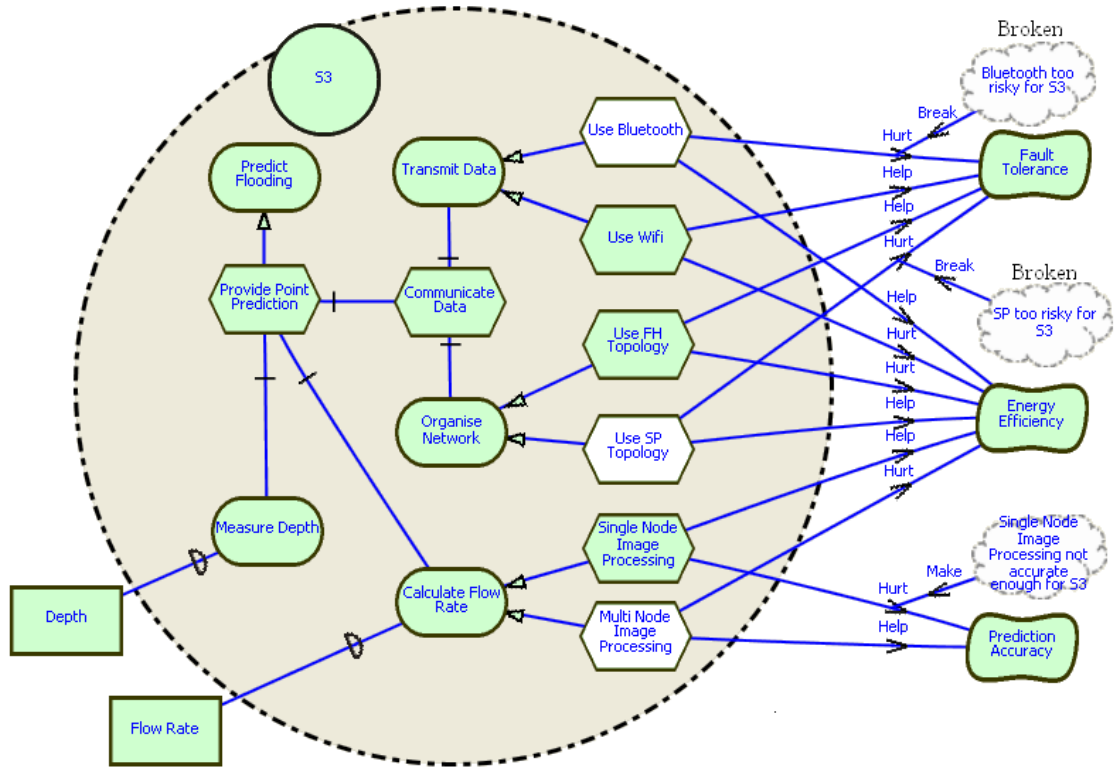


Figure 68: More Extensively Modified GridStix  $S_3$  Strategic Rationale Model

As with the previous example, Figure 67 shows the “Broken” label having been applied to the “Faults Likely in  $S_3$ ” claim, and in this case having propagated to two bottom-level claims: “SP too Risky for  $S_3$ ” and “Bluetooth too Risky for  $S_3$ ”. Figure 68 shows that the two bottom-level claims’ contributions to the Strategic Rationale model have been inverted, significantly adjusting the specification for the  $S_3$  target system to use less power given the reduced threat of node failure. Unsurprisingly, a simulator scenario in which there are few or no node failures whilst  $S_3$  is active, indicates that the less fault tolerant yet more energy efficient configuration reached after model modification is preferable.

To conclude, this section has demonstrated how the model transformer presented in Section 6.3 could modify the models of a larger DAS at run time in response to events raised by assumption monitors. The availability of a GridStix simulator has allowed the section to verify that the model modifications and resultant changes to target system



specifications are indeed beneficial under the circumstances under which they would occur. This ability to depart from the original system specification in response to unexpected operating conditions represents a step forward in the state of the art with respect to DASs, with previous DASs possessing only an ability to select between explicitly pre-specified configurations, which are of course dependent on developers having predicted operating conditions accurately.

It is debatable whether a m-DAS capable of adopting a finite number of configurations not codified as target system designs is truly devising a new configuration. Likewise, the act of identifying and modelling assumptions, and monitoring them could be considered to be affording some consideration to a set of circumstances a m-DAS may encounter outside of the originally envisaged domain, particularly if those conditions are then tested. However, a GridStix m-DAS as discussed in this section would certainly have a wider operational envelope than its (non model-driven) DAS counterpart, and would certainly be less tied to rigid domain partitioning than the original system. Thus, a suitably constructed m-DAS offers an opportunity to create a less brittle autonomous system.

Of course, granting a system any degree of autonomy removes a proportionate degree of assurance over its behaviour. A standard DAS (non m-DAS) guarantees that it will always be in one of a set number of pre-specified configurations, - even if there is a risk of the DAS being in an undesirable configuration under some circumstances. A m-DAS, however, offers no such guarantee in that one or more of the configurations may have been modified in response to monitoring data. Thus, the issue of validating m-DAS behaviour becomes increasingly important. The next section discusses how the possibility of emergent behaviour as a result of modification can be explored and mitigated.

## 7.7 Testing for Emergent Behaviour

The limited ability of a m-DAS to tailor its operation to operating conditions outside of those envisaged by developers presents an opportunity to develop systems able to operate

in more volatile, or less well understood environments than even standard DASs. However, granting this additional autonomy removes the ability of the developer to be completely sure of the system's behaviour under all operating contexts. For systems requiring a good degree of assurance, this combination of environmental and behavioural uncertainty proves problematic.

Section 7.6 demonstrates that a version of the GridStix DAS retrofitted with model-driven adaptive capability would be better equipped to operate in contexts in which the assumptions underpinning its design no longer hold. However, in circumstances where the m-DAS has adjusted its behaviour outside the original specification, it is very difficult to offer assurance that the m-DAS's behaviour will be appropriate, yet alone correct or optimal.

Two potential solutions that may offer a good degree of assurance without sacrificing a m-DASs flexibility is to use human monitoring, or human-in-the-loop adaptation (or at the very least human in-the-loop model modification). Unfortunately, the GridStix system operates in a remote location and communicates over a low bandwidth cellular link. Thus, a human in the loop would be infeasible, and the amount and detail of monitoring data that can be transmitted, coupled with the inherent delay in burst-mode transmission, would render monitoring ineffectual. Thus, the GridStix developers must rely on testing to deliver assurance.

The GridStix system is something of a borderline case with regards to classification as a safety-critical system. The environment agency and the met. office have other mechanisms in place to predict and warn of floods, meaning that life and limb are not directly or exclusively reliant on the system. However, it is trivial to imagine a scenario in which a failure of the GridStix system to deliver a timely flood warning would negatively impact the ability of those affected to evacuate and escape the rising water. Regardless of whether the GridStix system may be formally classified as safety-critical, it is clear that the developers would like to maximise the degree of assurance offered.

Although the GridStix system's individual components, the specified target systems and the GridStix system's adaptive behaviour all need to be tested, the testing required by a standard DAS is assumed. The section focusses instead on the additional testing required to uncover emergent behaviour in the m-DAS as a result of model modification. As discussed in Section 6.4, testing for emergent behaviour as a result of model modification involves devising test cases for all combinations of held and broken bottom-level claims, in much the same way as validation scenarios are devised. Thus, as with the validation scenarios in Section 7.5, the Level-One Claim Refinement Model for GridStix's  $S_3$  target system (depicted in Figure 49 on page 149) is analysed.

The three bottom level claims depicted in Figure 49 would yield seven combinations for which testing scenarios need be devised. A table of the individual combinations was presented as Table 5 on page 169 for reference. As with the validation scenarios, the explosive increase in required test cases as the number of bottom-level claims rises means that some way of prioritising the most likely to be encountered test cases is beneficial, and claim refinement models are again annotated with confidence labels.

However, when dealing with m-DAS test cases, the meaning of the “Unbreakable” label is changed. Previously, when dealing with validation scenarios, the “Unbreakable” has been lent to claims in which full confidence is held. For m-DAS test cases, “Unbreakable” label refers to the m-DAS having no means to monitor the claim. Thus, only claims which are unmonitorable (or merely un-monitored) and cannot be invalidated through propagation are labelled “Unbreakable”. Monitored claims, even if axiomatic, are classified either as “Qualified” or “Uncertain”, depending on the degree of confidence held. In borderline cases, the risk of claim falsification may be considered, with claims supporting large branches of the model classified “Uncertain”.

Figure 69 depicts the Claim Refinement Model for the  $S_3$  target system with confidence labels for test case consideration. Claims the m-DAS can monitor directly are labelled “Monitorable”, and have a certainty label applied. The certainty labels are propagated throughout the model, and the remaining claims labelled as “Unbreakable”.

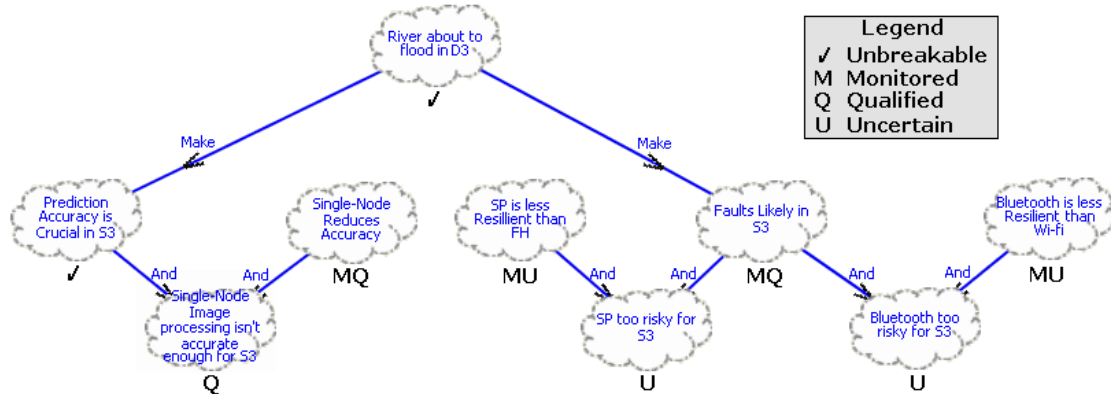


Figure 69: GridStix  $S_3$  Claim Refinement Model with Testing Claim Classification

The annotations in the model depicted in Figure 69 differ slightly from those in the model depicted in Figure 63 on page 170, as a result of the difference in criteria for labelling claims “Unbreakable”. In Figure 63, the “Faults Likely in  $S_3$ ” claim was labelled “Unbreakable”, given that a flooding river is certain to render GridStix nodes inoperable whilst submerged, and the additional risk of causing node damage either through water damage or due to debris. In Figure 69, however, the m-DAS's ability to monitor the number of faults encountered whilst the  $S_3$  target system is active means that the claim is labelled “Qualified”.

To devise testing scenarios to uncover emergent behaviour as a result of a m-DAS performing model modification, it is necessary to provide testing coverage for all combinations of invalidated bottom-level claims, in a manner similar to the way identification of validation scenarios was carried out for the system in Section 7.5. When creating validation scenarios, ideally all combinations would be considered, with unbreakable and, if necessary, qualified claims excluded solely to reduce validation burden. However, when devising testing scenarios, unbreakable claims can safely be omitted in all cases, given that the m-DAS has no ability to monitor and thus invalidate unbreakable claims. Thus, to offer complete testing coverage of the results of model modification, it is necessary to devise test cases covering all combinations of qualified and uncertain bottom level claims being invalidated or holding.

The number of test cases required can be calculated in the same way as the number of validation scenarios required for a given number of bottom-level claims was calculated. The number of test cases ( $T$ ) can be found from the number of bottom-level claims ( $n$ ) using the formula:  $T=2^n-1$ . One is subtracted here, because the combination of claims in which all hold is the original target system design, without any model modification having occurred. Thus, seven testing scenarios are needed to explore the  $S_3$  target system for possible emergent behaviour as a result of model modification. In cases in which providing testing coverage for all combinations of claims is too burdensome, it is possible to exclude the bottom-level claims classed as qualified, at the expense of it being possible for the m-DAS to adopt a completely untested configuration in some circumstances. This exclusion also introduces the risk of inaccuracy in claim classification: a bottom-level claim classed as qualified (either directly or using propagation) that is later invalidated would mean that the m-DAS may adopt an untested configuration, when a more cautious categorisation would have meant the configuration was tested.

Table 8 below depicts the number of testing scenarios required by all of the GridStix target systems, testing either all combinations of qualified and uncertain bottom-level claims, or just combinations of uncertain bottom-level claims. The “All Claims” column is included to illustrate the benefit of adjusting the criteria for “Unbreakable” claim classification, as without this adjustment, they too would need to be considered when devising test cases.

Target System	All Claims	Excluding Unbreakable Claims	Excluding Qualified & Unbreakable Claims
$S_1$	31	7	0
$S_2$	7	3	1
$S_3$	7	7	3
<b>Total</b>	45	17	4

*Table 8: GridStix Test Case Reduction using Claim Classification*

Table 8 shows that excluding qualified claims from test case generation offers a significant reduction in testing burden, whilst still testing the configurations adopted in cases where the riskier claims are invalidated. Focussing testing effort on the model modifications most likely allows limited testing resources to be directed appropriately, albeit risking the possibility of a m-DAS adopting an untested configuration. Even uncertain claims are expected to hold, as assumptions are expected to be validated as understanding of the environment and system components improves, which means that some may find this risk acceptable. Devising test cases to cover all combinations of qualified and uncertain claims, however, allows a m-DAS's additional adaptive capability to be fully tested.

The differences between the figures presented in Table 8 and those in Table 7 on page 172 are again due to the change in criteria governing the classification of claims as “Unbreakable”.

To conclude, this section uses a detailed case study to illustrate how test cases designed to uncover emergent behaviour as a result of model modification can be constructed for a m-DAS. The section highlights the differences between the claim classification used in requirements validation and testing, and shown how this adjustment reduces the number of test cases required. The section demonstrates how this number can be further reduced by excluding qualified claims from consideration, but has discussed the risk inherent in doing so.

As discussed in the previous section, the act of performing testing to verify a m-DAS's behaviour after all potential model modifications means that a key m-DAS's benefit: tailoring behaviour to unforeseen operating conditions, is removed. If the m-DAS has been tested under a set of conditions, just how unforeseen can they actually be? Of course, if testing scenarios are omitted, for example after using the test case reduction method discussed in this section, there are still some sets of conditions under which the m-DAS's behaviour is less certain. However, the GridStix m-DAS certainly offers a wider operational envelope than the original system, and this section demonstrates circumstances in which this could prove beneficial.

## 7.8 Chapter Conclusion

The chapter demonstrates the production and use of ReAssuRE models for a substantial, real life DAS. The case study confirms it possible to capture real-life rationale, which may be nuanced or contradictory and overlapping several related decisions, using claims. Furthermore, the case study shows that relatively complex, balanced, real-life rationale can, once captured, be used to better manage change; which may affect a DAS's specification due to actual changes in the environment or goals, or as understanding of the originally envisaged environment and DAS improves.

The availability of a GridStix simulator, allowing the performance of the GridStix system with different adaptation policies to be assessed under controlled environmental conditions, serves as an asset in validating the outcomes requirements changes effected using traceability information from ReAssuRE models. Given that the discussed requirement changes were scenario based, the ability to verify that configuration changes arrived at, and potentially steered towards by ReAssuRE models are indeed beneficial in the scenario detailed means that this chapter has demonstrated the effectiveness of a change process using tracing information from the models.

The case study allows the adaptation policy derivation method, and associated generation tool discussed in Section 5.2 to be validated by comparing the policies created by the tool with those originally created by hand for the DAS during development. Verifying this ability is important, given the reliance of the m-DAS architecture discussed in Section 6.3 on the policy derivation method, which is used to derive new target system configurations after model modification.

The case study offers significant insight into the utility of ReAssuRE models in identifying areas of uncertainty in a DAS's specification, and in devising ways in which the consequences of the uncertainty on the DAS's operation can be explored using validation scenarios. The case study identifies a limitation in the usefulness of validation scenarios, after it was noted that in some circumstances, it is easier to undertake effort on

validating an assumption codified in a claim than it is to explore the behaviour of the DAS should the assumption prove erroneous.

The case study's exploration of the potential benefits of granting the GridStix system more autonomy over its operation by converting it to a m-DAS moves beyond the capability implemented in the deployed system. However, the case study allows both the utility and potential risk of allowing a system to devise its own configurations in limited circumstances outside those envisaged at design time to be assessed.

The case study allows the m-DAS emergent behaviour test-case identification method discussed in Section 6.4 to be examined, with the potentially burdensome amount of required testing emerging as a significant obstacle. In mitigation, the test case reduction method also discussed in Section 6.4 is shown to offer a significant reduction in this burden by discarding unnecessary test-cases, and an even larger reduction possible if the level of assurance required is dropped, and the risk of claim mis-classification acceptable.



## 8 Conclusions and Future Work

---

This thesis examines the use of rich tracing information in requirements models for dynamically adaptive systems. Such tracing information can be used to manage change, which this thesis has argued will be prevalent given the uncertain, volatile and often novel nature of DASs and their operating environments. This thesis has also demonstrated how the captured tracing information can be used to identify areas of uncertainty in understanding of the environment or expected system behaviour, with a view to either reducing or mitigating the uncertainty.

This thesis also demonstrates how the same tracing information could be used by a DAS itself at run time, to better adapt to unforeseen operating conditions. This run-time use of tracing information captured in design-time requirements models is not only a novel concept, but offers the potential to create DASs with a greater degree of autonomy than currently possible.

This chapter starts by returning to the research questions posed at the beginning of the thesis, in Section 1.3. The answers to the questions which have emerged throughout the thesis are presented and discussed. The chapter then revisits the research hypothesis, presented in Section 1.4, examining the results of the case study presented in Chapter 7 to see whether they support the original hypothesis. The chapter then discusses the limitations of the research, before moving on to discuss the possible future work opened up by the research. Finally, the chapter offers some closing remarks, drawing the thesis to a close.

### 8.1 Research Questions Revisited

This section revisits the research questions presented in Section 1.3, and discusses the answers to each, in turn, that have emerged throughout the thesis.

The first research question is:

To what extent is a DAS's adaptive behaviour merely a derivation of environmental analysis and configuration decisions?

Section 5.2 presents and discusses an adaptation policy generator, capable of deriving adaptation policies compatible with the GridKit [39] middleware from ReAssuRE models. Adaptation policies, as discussed in Section 2.4.1, prescribe the adaptive behaviour of DASs built using policy-driven adaptive middleware. ReAssuRE Level-One Strategic Rationale and Level Two models codify the results of environmental analysis undertaken during the RE process and the outcome of decisions based upon this analysis. Thus, the ability of the policy generation tool to derive adaptation policies controlling DAS adaptive behaviour indicates that the answer to this question is “entirely”.

Section 5.2 also presents a similar tool capable of generating specifications of DAS adaptive behaviour in the Genie Domain Specific Language (DSL) [107]. This tool allows other adaptive infrastructures, using incompatible policy formats, to use adaptive behaviour specifications derived from ReAssuRE models. This ability to generate middleware-independent adaptive behaviour serves to prove the generality of the adaptive behaviour derivation approach, lending additional credence to the claim that DAS adaptive behaviour is a mere derivation of environmental analysis and configuration decisions taken during the RE process.

The second research question is:

Can the information from the environmental analysis and configuration decisions be codified in models, and is it useful to do so?

It could be argued that the existing LoREM process [16], upon which ReAssuRE is based, had an existing ability to record configuration decisions, which are a result of environmental analysis. However, Section 4.2 demonstrates that this ability is limited in that the rationale behind the decisions, also a result of environmental analysis, remains implicit. ReAssuRE models use claims to record the rationale behind configuration

decisions, along with the alternatives considered, meaning that more of the information discussed in this research question is codified in ReAssuRE models. Sections 5.1.2 and 7.2 demonstrate the utility of this additional information, codified in model form, in performing model evolution. Sections 5.3 and 7.5 demonstrate the utility of this same information in performing requirements validation. Sections 6 and 7.6 demonstrate how the information could be used in model form by DASs at run time to adapt to unforeseen operating conditions. Thus, this research question can be answered positively.

The third research question is:

Given that both the information from the environmental analysis and the configuration decisions are subject to change, how can the workload of deriving a DAS's adaptive behaviour be reduced?

As discussed in answering the second research question, Sections 5.1.2 and 7.2 demonstrate how the improved traceability of ReAssuRE models enhance changeability. Improving the changeability of the requirements models from which a DAS's adaptive behaviour is derived can help to reduce the workload induced by a change. As for the workload involved in performing adaptive behaviour derivation specifically, Section 5.2 presents two tools capable of deriving adaptation policies or Genie DSL adaptive behaviour specifications, respectively. These tools can all but eliminate the workload associated with re-deriving DAS adaptive behaviour after models have been changed.

The fourth research question is:

How can a system be designed with a greater degree of autonomy than current state-of-the-art DASs, and is the extra autonomy useful?

Chapter 6 discusses how a Model-driven Dynamically Adaptive System (m-DAS) could be constructed using ReAssuRE models and assumption monitoring. Such a system would have the ability to adopt configurations not explicitly specified at design time when faced with one or more modelled assumptions that no longer hold. Section 7.6 demonstrates

how this extra autonomy could prove useful should the understanding upon which individual target system designs were based prove to be incorrect at run time.

The fifth and final research question is:

How can the testing workload be managed in systems with greater autonomy?

The answer to this final research question is less conclusive. Sections 6.4 and 7.7 demonstrated a method by which limited testing resources could be directed to the most likely of testing scenarios that have the potential to uncover emergent behaviour. However, the degree of assurance it is possible to afford to a system capable of adopting unexpected and untested configurations will inevitably be reduced. Other means of delivering assurance such as monitoring or human-in-the-loop adaptation, or human-in-the-loop model evolution, may prove a more acceptable solution.

## 8.2 Research Hypothesis Revisited

This section revisits the research hypothesis presented in Section 1.4, which is repeated below:

When performing early-phase RE for a Dynamically Adaptive System, the recording of additional tracing information will better support change later in the software engineering process. Recorded tracing information can be used during development to derive the adaptive behaviour of a DAS (the concern of switching from one configuration to another as the environment changes), and by the DAS itself after deployment to better adapt to unforeseen conditions.

The first component of the hypothesis concerns DAS requirements change. This thesis has demonstrated how recording additional tracing information better supports this change, with case study examples presented in Sections 5.1.2 and 7.2. As discussed in Section 5.1.1, some classes of DAS requirements change are indeterminate in their tracing requirements. Therefore, it is of course impossible to determine that *all* change is better

supported by ReAssuRE models' additional tracing information. However, this thesis demonstrates that *some* changes are better supported by the recording of additional traceability information, demonstrating the approach's utility.

The second component of the hypothesis concerns the use of tracing information codified in ReAssuRE models to derive DAS adaptive behaviour. Sections 5.2 and 7.3 of this thesis demonstrates an ability to derive adaptation policies directly or to derive adaptive behaviour specified in an appropriate Domain Specific Language in Sections 5.2 and 7.3. Thus, this clause of the hypothesis is supported by the research presented in this thesis.

The final clause of the hypothesis concerns the use of tracing information codified in ReAssuRE models to guide DAS adaptation at run time. This thesis demonstrates a method by which a new type of DAS, a m-DAS, which uses design-time requirements models to guide run-time adaptation. Section 6.3 presents an architecture by which a m-DAS may be constructed using ReAssuRE models and assumption monitoring techniques. Section 6.3 also presented a proof-of-concept model transformer component that demonstrates the viability of such a system. The ability of a m-DAS using the described architecture and the proof of concept model transformer to better adapt to unforeseen conditions is demonstrated in Section 7.6. Thus, this clause of the hypothesis is supported by the research presented in this thesis.

Section 1.4 describes the method used to conduct the research presented in this thesis. The wide and varied nature of the Software Engineering discipline hinders the emergence of a single accepted research method [12]. The synthetic nature of this thesis's more specific field: requirements modelling, lends the use of case studies as an appropriate validation tool. Chapter 7 makes use of simulation to more objectively validate this thesis's claimed benefits within the more detailed case study. The outcomes of the two case studies, taken complete with the simulator results where appropriate, lead to an overall conclusion that the hypothesis is substantially supported.

## 8.3 Limitations

This section highlights the limitations of the research, along with some reflective discussion of the limitations and possible mitigation.

Perhaps the most significant limitation of the work is the requirement for an operating environment to be partitioned into distinct domains. This requirement was inherited from the LoREM method upon which ReAssuRE is based [16], and in turn the original RE perspectives defined in [10]. Although this thesis is clear that ReAssuRE's applicability extends only to systems whose operating environments may be partitioned, there exists no research defining whether all, some, or few proposed DAS environments meet this criterion. One potential avenue of future research lies in the analysis of proposed or implemented DASs to establish the proportion of DAS operating environments for which approaches requiring domain partitioning would be appropriate.

Another important limitation is inherited from the  $i^*$  modelling ontology itself [71]. The complexity of  $i^*$  models as scale increases is a known concern [104] [105] [106], and although  $i^*$  modelling tools feature conditional display tools allowing modellers to work at varying degrees of abstraction [84] [73], the problem remains largely unsolved. Thus, ReAssuRE models which require additional model elements to record rationale, are subject to the problem to an even greater degree.

The separation of Claim Refinement Models from Strategic Rationale at Level One limits the additional model elements on Strategic Rationale models to bottom-level claims. However, some would argue that the two Level-One models should be combined, removing the duplication of bottom-level claims between the two. Although possible, the significant number of additional model elements placed on the Strategic Rationale model; already acknowledged to suffer from clutter, means that this approach is rejected.

Section 4.2.1 discusses the trade off that can be made between the maximisation of the detail and precision of recorded rationale versus the minimisation of the number of claims required to record it. The decision taken by the modeller on the balance struck between

these competing objectives has a clear impact on the complexity of the completed ReAssuRE models. Thus, the ReAssuRE method possesses a degree of flexibility which can be leveraged to manage the problem of model complexity.

In much the same way as there is a tension between the precision of recorded rationale and the complexity of the model in which it is recorded, there is also a tension between the number of alternatives to consider and model versus the complexity of the model in which they are recorded. This thesis does not prescribe a set number of alternatives to consider for a given decision, and as mentioned for the precision of rationale, the ReAssuRE method possesses a degree of flexibility in that it does not depend on a specific number of alternatives having been considered. It should be noted, however, that neither case study in the thesis had a large number of alternatives for a single decision. Thus the value of the tracing information recorded has not been compared to the modelling effort required to record more than a handful of alternatives for a single decision.

As mentioned briefly in Section 8.2, and discussed more completely in Section 5.1.1, the indeterminate requirements of some classes of DAS requirements change prevents a requirements modelling method from claiming complete support for all requirements changes. This limitation appears insurmountable, and would be shared by any competing approach. However, Sections 5.1.2 and 7.2 demonstrate that ReAssuRE models support the tracing needs to efficiently support several classes of change a DAS's requirements may be subjected to.

The burden of requirements validation when devising systems to operate in uncertain, volatile environments is always going to be significant. The validation scenario reduction method presented in Section 5.3 allows redundant validation activity to be identified and omitted, and allows the amount of remaining validation activity to be prioritised. However, this method mitigates rather than overcomes the validation burden. Likewise, the m-DAS emergent behaviour test case reduction method discussed in Section 6.4 offers only a potential reduction in testing burden in exchange for a reduction in delivered assurance. This thesis, however, identifies other means of delivering m-DAS assurance which may prove more appropriate in some cases.

There is a final limitation that lies within the research method used. The use of a case study is insufficient to prove a hypothesis, merely to demonstrate that it holds in some circumstances. Thus, the use of a case study to validate the research presents a threat to validity in terms of generality. Typically, methods are best proved beneficial by empirical research over time. Unfortunately, the immature nature of DASs as a class of system means that there is little opportunity to conduct such empirical research. Thus, although this thesis can conclude that the research presented supports its stated hypothesis, no stronger conclusion can accurately be reached.

## 8.4 Future Work

In addition to supporting or disproving the thesis's hypothesis. The research presented in this thesis allows new, previously unidentified areas of research to emerge. Such research would either enhance the support of this thesis's hypothesis or allow the research to be further extended delivering greater or wider benefit. This section details several of the identified areas in which future research would be beneficial.

The most promising avenues of future research stem from the Model-driven Dynamically Adaptive Systems (m-DASs) discussed in Chapter 6 and demonstrated in Section 7.6. a m-DAS possesses a limited ability to tailor its operation to conditions outside those envisaged *a priori*, which is a novel capability. The m-DAS architecture presented in Section 6.3 uses ReAssuRE models and assumption monitoring, which is in itself a novel re-purposing of requirements monitoring [30] [95] techniques.

There is currently significant research interest in the use of requirements models at run time [112] or some other representation of system requirements [97]. Furthermore, the potential, and consequences, of a system capable of tailoring its operation to conditions not envisaged at design time, using any means, is relatively unexplored. The construction of a m-DAS using the architecture detailed in Section 6.3 would allow research in these areas to be conducted effectively.



Although this thesis demonstrates a proof of concept for a m-DAS constructed using ReAssuRE models as run-time representations of DAS requirements and using assumption monitoring to drive model transformation, it may be possible to construct a m-DAS using other means. Other run-time requirements representations, or other means of performing model transformation may exhibit characteristics different from those exhibited by m-DASs constructed as described in this thesis. Thus, research could be conducted into alternate means of constructing m-DASs.

As mentioned in the previous section, the immature nature of DASs as a class of system means empirical data on their performance, effectiveness, and suitability for different types of operating environment is scarce. For the novel m-DAS, this scarcity is exaggerated. Thus, there is a significant need for empirical research into deployed, real-world, DASs and m-DASs.

As mentioned earlier in this section, the application of requirements monitoring techniques to assumptions is, in itself, novel. Although this thesis makes no claim as to the utility of assumption monitoring in a system other than a m-DAS, there is a possibility of the technique being used in non-adaptive systems. In such systems, assumption monitoring could be used in a manner similar to requirements monitoring, to better inform stakeholders of operating conditions outside those the system was designed for, and to better inform maintenance and tuning activity of the experienced operating environment. Research could thus be performed examining the use of assumption monitoring data in performing research and tuning.

Section 8.3 identified a limitation to the research in that there exists no research establishing the number or proportion of adaptive system operating environments that can be suitably partitioned for ReAssuRE or related approaches. If few operating environments can be partitioned into domains, ReAssuRE and related approaches solve a niche problem. If more or most environments may be suitably partitioned, the case studies presented in this thesis may more closely resemble the general case. Thus, research is required into adaptive system operating environments.

## 8.5 Closing Remarks

This thesis explores the use of tracing information in requirements models to increase the utility of the models later in the requirements engineering process, later in the software engineering process, and at run time. The tracing information can be used later in the RE process to effect requirements change efficiently, or to identify areas of uncertainty in the analyst's understanding of the system or environment. At run time, a suitably constructed m-DAS can use the tracing information to adapt to contexts beyond those envisaged at design time. Used later in the SE process, the tracing information can be used to derive a system's adaptive behaviour, or to identify key testing scenarios for m-DASs.

This thesis presents the ReAssuRE modelling process, which defines modelling perspectives from which a DAS is viewed, and prescribes the recording of tracing information that can be used as described above. Also presented is a proof of concept for the m-DAS class of system, which interprets ReAssuRE models at run time, transforming them in response to monitored conditions and deriving new adaptive behaviour when appropriate.

The novel m-DAS class of system has the potential to allow the construction of systems offering a greater degree of autonomy than currently possible, which may well prove to be something of a double-edged sword. Systems capable of a greater degree of autonomy may prove suitable for operation in more volatile, more uncertain and less well understood environments. However, systems capable of greater autonomy are far more challenging to deliver assurance in, and thus to depend upon. The two edges of this metaphorical sword represent the areas in which this thesis aims to drive further research.

## Appendix A – Image Viewer's Adaptation Policy

---

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <ReconfigurationRules>
    <ReconfigurationRule>
      <FrameWork>Cache</FrameWork>
      <Events>
        <Event>
          <Type>HIGH_LATENCY</Type>
          <Value/>
        </Event>
      </Events>
      <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.Cache</Name>
      </Reconfiguration>
    </ReconfigurationRule>
    <ReconfigurationRule>
      <FrameWork>Cache</FrameWork>
      <Events>
        <Event>
          <Type>LOW_LATENCY</Type>
          <Value/>
        </Event>
      </Events>
      <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.No_Cache</Name>
      </Reconfiguration>
    </ReconfigurationRule>
  </ReconfigurationRules>
```

## Appendix B – GridStix Adaptation Policy

---

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <ReconfigurationRules>
    <ReconfigurationRule>
      <FrameWork>SpanningTree</FrameWork>
      <Events>
        <Event>
          <Type>HIGH_FLOW</Type>
          <Value>TRUE</Value>
        </Event>
      </Events>
      <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.MultiNode</Name>
      </Reconfiguration>
    </ReconfigurationRule>
    <ReconfigurationRule>
      <FrameWork>SpanningTree</FrameWork>
      <Events>
        <Event>
          <Type>HIGH_FLOW</Type>
          <Value>TRUE</Value>
        </Event>
      </Events>
      <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.Wifi</Name>
      </Reconfiguration>
    </ReconfigurationRule>
    <ReconfigurationRule>
      <FrameWork>SpanningTree</FrameWork>
      <Events>
        <Event>
          <Type>FLOODPREDICTED</Type>
          <Value>TRUE</Value>
```

```

        </Event>
    </Events>
    <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.FewestHop</Name>
    </Reconfiguration>
</ReconfigurationRule>
<ReconfigurationRule>
    <FrameWork>SpanningTree</FrameWork>
    <Events>
        <Event>
            <Type>FLOODPREDICTED</Type>
            <Value>TRUE</Value>
        </Event>
    </Events>
    <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.Wifi</Name>
    </Reconfiguration>
</ReconfigurationRule>
<ReconfigurationRule>
    <FrameWork>SpanningTree</FrameWork>
    <Events>
        <Event>
            <Type>HIGH_FLOW</Type>
            <Value>FALSE</Value>
        </Event>
    </Events>
    <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.SingleNode</Name>
    </Reconfiguration>
</ReconfigurationRule>
<ReconfigurationRule>
    <FrameWork>SpanningTree</FrameWork>
    <Events>
        <Event>
            <Type>HIGH_FLOW</Type>

```

```

        <Value>FALSE</Value>
    </Event>
</Events>
<Reconfiguration>
    <FileType>Java</FileType>
    <Name>Reconfigurations.Bluetooth</Name>
</Reconfiguration>
</ReconfigurationRule>
<ReconfigurationRule>
    <FrameWork>SpanningTree</FrameWork>
    <Events>
        <Event>
            <Type>FLOODPREDICTED</Type>
            <Value>TRUE</Value>
        </Event>
    </Events>
    <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.SingleNode</Name>
    </Reconfiguration>
</ReconfigurationRule>
<ReconfigurationRule>
    <FrameWork>SpanningTree</FrameWork>
    <Events>
        <Event>
            <Type>FLOODPREDICTED</Type>
            <Value>TRUE</Value>
        </Event>
    </Events>
    <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.FewestHop</Name>
    </Reconfiguration>
</ReconfigurationRule>
<ReconfigurationRule>
    <FrameWork>SpanningTree</FrameWork>
    <Events>
        <Event>

```

```

        <Type>HIGH_FLOW</Type>
        <Value>FALSE</Value>
    </Event>
</Events>
<Reconfiguration>
    <FileType>Java</FileType>
    <Name>Reconfigurations.ShortestPath</Name>
</Reconfiguration>
</ReconfigurationRule>
<ReconfigurationRule>
    <FrameWork>SpanningTree</FrameWork>
    <Events>
        <Event>
            <Type>HIGH_FLOW</Type>
            <Value>FALSE</Value>
        </Event>
    </Events>
    <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.Bluetooth</Name>
    </Reconfiguration>
</ReconfigurationRule>
<ReconfigurationRule>
    <FrameWork>SpanningTree</FrameWork>
    <Events>
        <Event>
            <Type>FLOODPREDICTED</Type>
            <Value>FALSE</Value>
        </Event>
    </Events>
    <Reconfiguration>
        <FileType>Java</FileType>
        <Name>Reconfigurations.MultiNode</Name>
    </Reconfiguration>
</ReconfigurationRule>
<ReconfigurationRule>
    <FrameWork>SpanningTree</FrameWork>
    <Events>

```

```
<Event>
  <Type>FLOODPREDICTED</Type>
  <Value>FALSE</Value>
</Event>
</Events>
<Reconfiguration>
  <FileType>Java</FileType>
  <Name>Reconfigurations.ShortestPath</Name>
</Reconfiguration>
</ReconfigurationRule>
</ReconfigurationRules>
```

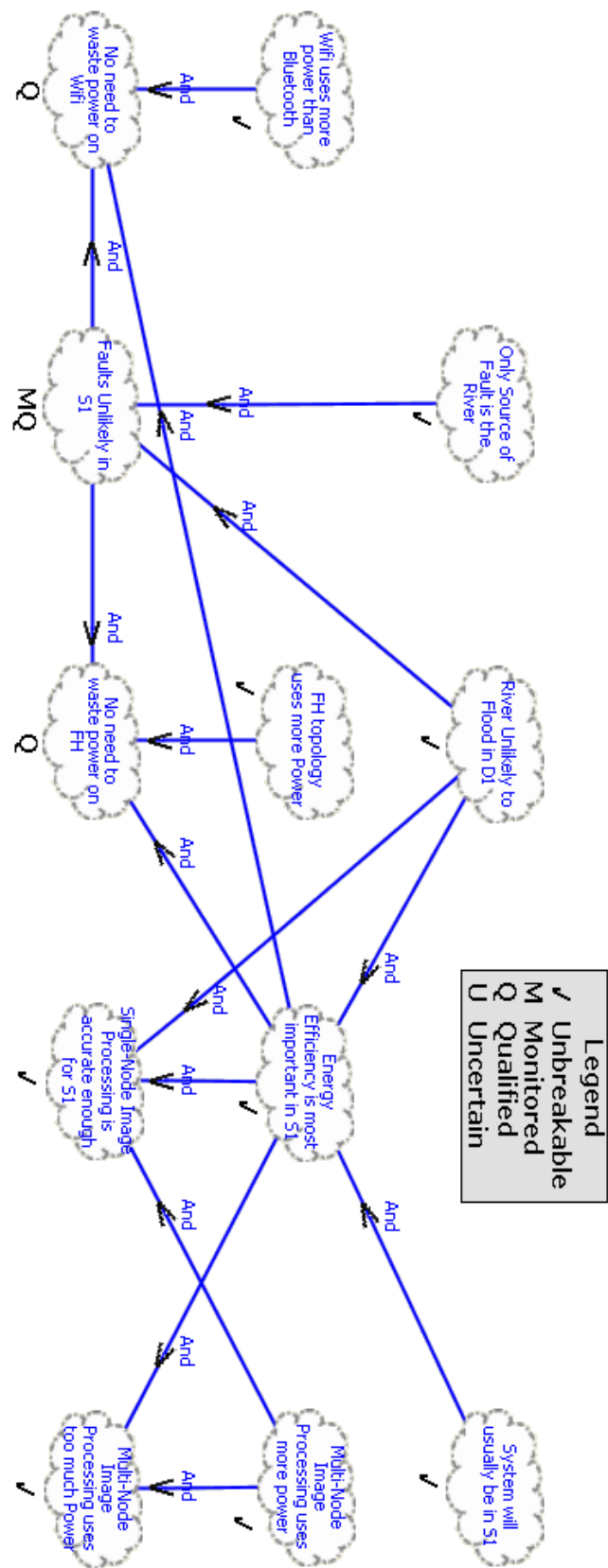


## **Appendix C – Annotated GridStix Claim Refinement Models**

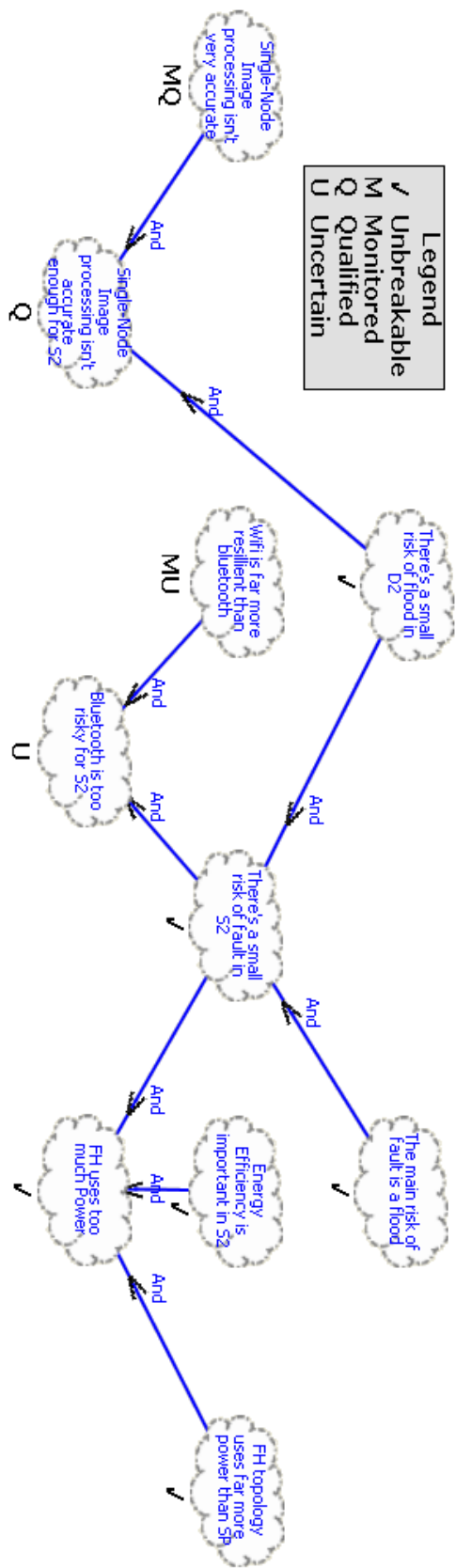
---

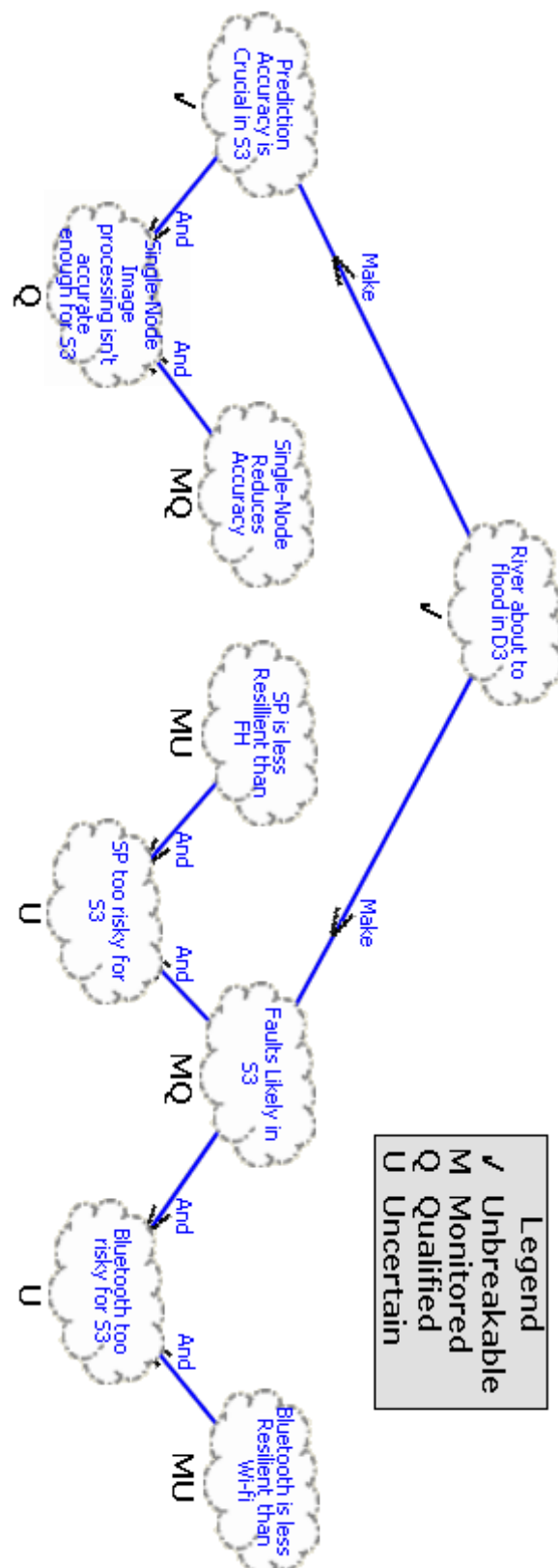
This appendix contains the Level-One Claim Refinement Models for the  $S_1$ ,  $S_2$  and  $S_3$  GridStix target systems, annotated with the monitoring and certainty labels used in the test-case identification and reduction methods.

S<sub>1</sub>



S<sub>2</sub>



$S_3$ 

## Appendix D – Pseudocode for Policy Generation

---

```

list VariationPoints = list()
for each Goal that appears in all Level-One SR models {
    if ( count(tasks are connected to the goal with means-end links) > 1 ) {
        VariationPoints.Add(Goal)
    }
}
list Reconfigurations = list(SourceTS, DestTS, RemoveComponent, AddComponent)
for each valid target system transition {
    for each VariationPoint in VariationPoints {
        if (pickedTask(sourceTS,VariationPoint) != pickedTask(destTS,VariationPoint)
        {
            Trigger = getTrigger( getLevel2(SourceTS, DestTS) )
            writeRule(
                pickedTask(sourceTS,VariationPoint),
                pickedTask(destTS,VariationPoint),
                Trigger
            )
        }
    }
}

function pickedTask(TargetSystem, Goal) {
    for each Claim in TargetSystem.getClaims() {
        for each Task in TargetSystem.getTasks() {
            if ( isLinked(TargetSystem,Claim,Task) ) && ( supports(Claim,Task) )
                { return Task; }
        }
    }
}

function getTrigger(Level2Model) {
    FireEventTask = Level2Model.findTask("Fire Event When Environment Changes");
    List PotentialTriggers = FireEventTask.getConnections();
    for each PotentialTrigger in PotentialTriggers {
        if ( PotentialTrigger.startsWith("Fire") ) { return PotentialTrigger; }
    }
}

supports function returns true if claim has a "make" link to a positive
contribution link attached to specified Task, or a "break" to a negative
writeRule function writes a rule for the specified component switch and trigger

```

## References

---

1. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing, (2003).
2. Hughes, D., Greenwood, P., Coulson, G., Blair, G.: GridStix: Supporting Flood Prediction using Embedded Hardware and Next Generation Grid Middleware. Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks. pp. 621-626 IEEE Computer Society (2006).
3. Kramer, J., Magee, J.: Self-Managed Systems: an Architectural Challenge. 2007 Future of Software Engineering. pp. 259-268 IEEE Computer Society (2007).
4. Welsh, K., Sawyer, P.: Managing Testing Complexity in Dynamically Adaptive Systems: A Model-Driven Approach. Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops. pp. 290-298 IEEE Computer Society (2010).
5. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. IEEE Computer. 42, 22-27 (2009).
6. Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., Wolf, A.L.: An Architecture-Based Approach to Self-Adaptive Software. IEEE Intelligent Systems. 14, 54-62 (1999).
7. Ghosh, D., Sharman, R., Rao, H.R., Upadhyaya, S.: Self-healing systems - survey and synthesis. Decis. Support Syst. 42, 2164-2185 (2007).
8. Trapp, M., Adler, R., Förster, M., Junger, J.: Runtime adaptation in safety-critical automotive systems. Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering. pp. 308-315 ACTA Press, Innsbruck, Austria (2007).
9. Cheng, B.H.C., Atlee, J.M.: Research Directions in Requirements Engineering. 2007 Future of Software Engineering. pp. 285-303 IEEE Computer Society (2007).
10. Berry, D.M., Cheng, B.H.C., Zhang, J.: The four levels of requirements engineering for and in dynamic adaptive systems. 11th International Workshop On

- Requirements Engineering Foundation For Software Quality (REFSQ). (2005).
11. Thomas, R.M.: Blending qualitative & quantitative research methods in theses and dissertations. Corwin Press (2003).
  12. Shaw, M.: The coming-of-age of software architecture research. Proceedings of the 23rd International Conference on Software Engineering. p. 656 IEEE Computer Society, Toronto, Ontario, Canada (2001).
  13. Owen, C.: Design Research: Building the Knowledge Base. 5, 36-45 (1997).
  14. Lapouchnian, A., Liaskos, S., Mylopoulos, J., Yu, Y.: Towards requirements-driven autonomic systems design. SIGSOFT Softw. Eng. Notes. 30, 1-7 (2005).
  15. Perry, D.E., Sim, S.E., Easterbrook, S.: Case studies for software engineers. pp. 1045-1046 ACM, Shanghai, China (2006).
  16. Goldsby, H.J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Hughes, D.: Goal-Based Modeling of Dynamically Adaptive System Requirements. Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems. pp. 36-45 IEEE Computer Society (2008).
  17. McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.: Composing Adaptive Software, (2004).
  18. Schilit, B., Adams, N., Want, R.: Context-Aware Computing Applications. Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications. pp. 85-90 IEEE Computer Society (1994).
  19. Zhang, J., Yang, Z., Cheng, B., McKinley, P.: Adding safeness to dynamic adaptation techniques. IEE Seminar Digests. 2004, 17-21 (2004).
  20. Samimi, F.A., Mckinley, P.K., Sadjadi, S.M.: Mobile Service Clouds: A Self-Managing Infrastructure for Autonomic Mobile Computing Services.
  21. Hughes, D., Greenwood, P., Blair, G., Coulson, G., Smith, P., Beven, K.: An Intelligent and Adaptable Grid-based Flood Monitoring and Warning System.
  22. Li, Q., van der Schaar, M.: Providing Adaptive QoS to Layered Video Over Wireless Local Area Networks Through Real-Time Retry Limit Adaptation. IEEE Transactions on Multimedia. 6, 290, 278 (2004).
  23. Krief, F.: Self-aware management of IP networks with QoS guarantees. Int. J. Netw.

- Manag. 14, 351-364 (2004).
24. Canfora, G., Penta, M.D.: Testing Services and Service-Centric Systems: Challenges and Opportunities, (2006).
  25. Baresi, L., Ghezzi, C., Guinea, S.: Towards Self-healing Composition of Services. Contributions to Ubiquitous Computing. pp. 27-46 (2007).
  26. Baresi, L., Guinea, S., Pasquale, L.: Self-healing BPEL processes with Dynamo and the JBoss rule engine. International workshop on Engineering of software services for pervasive environments: in conjunction with the 6th ESEC/FSE joint meeting. pp. 11-20 ACM, Dubrovnik, Croatia (2007).
  27. Sierra, C., Faratin, P., Jennings, N.R.: A Service-Oriented Negotiation Model between Autonomous Agents. Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: Multi-Agent Rationality. pp. 17-35 Springer-Verlag (1997).
  28. Liu, Y., Ngu, A.H., Zeng, L.Z.: QoS computation and policing in dynamic web service selection. Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. pp. 66-73 ACM, New York, NY, USA (2004).
  29. Canfora, G., Penta, M.D., Esposito, R., Villani, M.L.: QoS-Aware Replanning of Composite Web Services. Proceedings of the IEEE International Conference on Web Services. pp. 121-129 IEEE Computer Society (2005).
  30. Fickas, S., Feather, M.: Requirements monitoring in dynamic environments. Requirements Engineering, IEEE International Conference on. p. 140 IEEE Computer Society, Los Alamitos, CA, USA (1995).
  31. Kalbarczyk, Z.T., Iyer, R.K., Bagchi, S., Whisnant, K.: Chameleon: A Software Infrastructure for Adaptive Fault Tolerance. IEEE Trans. Parallel Distrib. Syst. 10, 560-579 (1999).
  32. Chang, I., Hiltunen, M.A., Schlichting, R.D.: Affordable fault tolerance through adaptation. Parallel and Distributed Processing, Lecture Notes in Computer Science 1388. 1388, 585--603 (1998).
  33. Welsh, K., Sawyer, P.: When to Adapt? Identification of Problem Domains for



- Adaptive Systems. Requirements Engineering: Foundation for Software Quality. pp. 198-203 (2008).
34. Robinson, D., Kotonya, G.: A Runtime Quality Architecture for Service-Oriented Systems. Proceedings of the 6th International Conference on Service-Oriented Computing. pp. 468-482 Springer-Verlag, Sydney, Australia (2008).
  35. Modafferi, S., Mussi, E., Pernici, B.: SH-BPEL: a self-healing plug-in for Ws-BPEL engines. Proceedings of the 1st workshop on Middleware for Service Oriented Computing (MW4SOC 2006). pp. 48-53 ACM, Melbourne, Australia (2006).
  36. Fickas, S.: Clinical requirements engineering. Proceedings of the 27th international conference on Software engineering. pp. 28-34 ACM, St. Louis, MO, USA (2005).
  37. Liaskos, S., Lapouchnian, A., Wang, Y., Yu, Y., Easterbrook, S.: Configuring Common Personal Software: a Requirements-Driven Approach. Proceedings of the 13th IEEE International Conference on Requirements Engineering. pp. 9-18 IEEE Computer Society (2005).
  38. Mozilla Foundation: Thunderbird - Reclaim your inbox, <http://en-gb.www.mozillamessaging.com/en-GB/thunderbird/>.
  39. Grace, P., Coulson, G., Blair, G., Mathy, L., Yeung, W.K., Cai, W., Duce, D., Cooper, C.: GRIDKIT: Pluggable Overlay Networks for Grid Computing. 1463--1481 (2004).
  40. Fraunhofer IESE: Welcome to the Ambient Assisted Living Environment, the Assisted Living Systems Laboratory!, [http://www.iese.fraunhofer.de/projects/med\\_projects/aal-lab/index.jsp](http://www.iese.fraunhofer.de/projects/med_projects/aal-lab/index.jsp).
  41. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H., Bruel, J.: RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. Requirements Engineering, IEEE International Conference on. pp. 79-88 IEEE Computer Society, Los Alamitos, CA, USA (2009).
  42. Denaro, G., Pezzé, M., Tosi, D., Schilling, D.: Towards self-adaptive service-oriented architectures. Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications. pp. 10-16 ACM, Portland, Maine (2006).
  43. Orriens, B., Yang, J.: A Rule Driven Approach for Developing Adaptive Service

- Oriented Business Collaboration. Proceedings of the IEEE International Conference on Services Computing. pp. 182-189 IEEE Computer Society (2006).
44. Gu, T., Pung, H.K., Zhang, D.Q.: A service-oriented middleware for building context-aware services. J. Netw. Comput. Appl. 28, 1-18 (2005).
  45. Tusch, R.: Towards an Adaptive Distributed Multimedia Streaming Server Architecture Based on Service-Oriented Components. Modular Programming Languages. pp. 78-87 (2003).
  46. Erradi, A., Maheshwari, P., Tosic, V.: Policy-driven middleware for self-adaptation of web services compositions. Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware. pp. 62-80 Springer-Verlag New York, Inc., Melbourne, Australia (2006).
  47. Ghezzi, C., Inverardi, P., Montangero, C.: Dynamically Evolvable Dependable Software: From Oxymoron to Reality. Concurrency, Graphs and Models. pp. 330-353 (2008).
  48. Blair, G., Coulson, G., Davies, N., Fitzpatrick, T., Yr, L.: Adaptive Middleware for Mobile Multimedia Applications. (1997).
  49. Schmidt, D.C.: Middleware for real-time and embedded systems. Commun. ACM. 45, 43-48 (2002).
  50. Sadjadi, S.M.: A Survey of Adaptive Middleware. Technical Report, Michigan State University (2003).
  51. Schmidt, D.C., Dept, C.E., Cross, J.K., Schantz, R.E., Sharp, D.C., Masters, M.W., Dipalma, L.P.: Towards Adaptive and Reflective Middleware For Network-Centric Combat Systems. (2001).
  52. Kon, F., Román, M., Liu, P., Mao, J., Yamane, T., Magalhã, C., Campbell, R.H.: Monitoring, security, and dynamic configuration with the *dynamicTAO* reflective ORB. IFIP/ACM International Conference on Distributed systems platforms. pp. 121-143 Springer-Verlag New York, Inc., New York, New York, United States (2000).
  53. Roman, M., Kon, F., Campbell, R.H.: Reflective Middleware: From Your Desk to Your Hand. IEEE Distributed Systems Online. 2, (2001).

54. Ledoux, T.: OpenCorba: A Reflektive Open Broker. Proceedings of the Second International Conference on Meta-Level Architectures and Reflection. pp. 197-214 Springer-Verlag (1999).
55. Coulson, G., Blair, G., Grace, P., Joolia, A., Lee, K., Ueyama, J.: A Component Model for Building Systems Software. in proc. IASTED software engineering and applications (SEA'04). (2004).
56. Lutfiyya, H., Molenkamp, G., Katchabaw, M., Bauer, M.A.: Issues in Managing Soft QoS Requirements in Distributed Systems Using a Policy-Based Framework. Proceedings of the International Workshop on Policies for Distributed Systems and Networks. pp. 185-201 Springer-Verlag (2001).
57. Lymberopoulos, L., Lupu, E., Sloman, M.: An Adaptive Policy-Based Framework for Network Services Management. J. Netw. Syst. Manage. 11, 277-303 (2003).
58. Ponnappan, A., Yang, L., Pillai, R., Braun, P.: A Policy Based QoS Management System for the IntServ/DiffServ Based Internet. Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02). p. 159 IEEE Computer Society (2002).
59. Sloman, M.: Policy driven management for distributed systems. Journal of Network and Systems Management. 2, 333-360 (1994).
60. White, S.R., Hanson, J.E., Whalley, I., Chess, D.M., Kephart, J.O.: An Architectural Approach to Autonomic Computing. Autonomic Computing, International Conference on. pp. 2-9 IEEE Computer Society, Los Alamitos, CA, USA (2004).
61. Weld, D.S.: Recent Advances in AI Planning. AI MAGAZINE. 20, 93--123 (1999).
62. Brian C. Williams, Urang Nayak: A model-based approach to reactive self-configuring systems, <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.1.6964>, (1996).
63. Muscettola, N., Nayak, P.P., Pell, B., Williams, B.C.: Remote Agent: To Boldly Go Where No AI System Has Gone Before. (1998).
64. Adve, V., Lam, V.V., Ensink, B.: Language and Compiler Support for Adaptive Distributed Applications. Proceedings of the 2001 ACM SIGPLAN workshop on Optimization of middleware and distributed systems. pp. 238-246 ACM, Snow Bird,

- Utah, United States (2001).
65. Kasten, E.P., Mckinley, P.K.: Adaptive Java: Refractive and Transmutative Support for Adaptive Software. (2001).
  66. Sadjadi, S.M., Mckinley, P.K., Cheng, B.H.C., Stirewalt, R.E.K.: TRAP/J: Transparent Generation of Adaptable Java Programs. (2004).
  67. Welch, I., Stroud, R.J.: Kava - A Reflective Java Based on Bytecode Rewriting. Proceedings of the 1st OOPSLA Workshop on Reflection and Software Engineering: Reflection and Software Engineering, Papers from OORaSE 1999. pp. 155-167 Springer-Verlag (2000).
  68. Morandini, M., Penserini, L., Perini, A.: Towards goal-oriented development of self-adaptive systems. Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems. pp. 9-16 ACM, Leipzig, Germany (2008).
  69. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems. 8, 203-236 (2004).
  70. Dardenne, A., Lamsweerde, A.V., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. 20, 3-50 (1993).
  71. Yu, E.S.K.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. 226--235 (1997).
  72. Yu, Y., Lapouchnian, A., Liaskos, S., Mylopoulos, J., Leite, J.C.S.P.: From Goals to High-Variability Software Design.
  73. OpenOME, an requirements engineering tool, <http://www.cs.toronto.edu/km/openome/>.
  74. Letier, E., Lamsweerde, A.V.: Deriving operational software specifications from system goals. Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering. pp. 119-128 ACM, Charleston, South Carolina, USA (2002).
  75. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time parameter adaptation. Proceedings of the 31st International Conference on Software Engineering. pp. 111-121 IEEE Computer Society (2009).

76. Yin, G., Zhang, Q.: Discrete-time Markov chains: two-time-scale methods and applications. Springer (2005).
77. Brown, G., Cheng, B.H.C., Goldsby, H., Zhang, J.: Goal-oriented specification of adaptation requirements engineering in adaptive systems. Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems. pp. 23-29 ACM, Shanghai, China (2006).
78. Goldsby, H., Cheng, B.H.C.: Goal-Oriented Modeling of Requirements Engineering for Dynamically Adaptive System. Proceedings of the 14th IEEE International Requirements Engineering Conference. pp. 338-339 IEEE Computer Society (2006).
79. Zhang, J., Cheng, B.: Using temporal logic to specify adaptive program semantics. Journal of Systems and Software. 79, 1361-1369 (2006).
80. Nakagawa, H., Ohsuga, A., Honiden, S.: Constructing Self-Adaptive Systems Using a KAOS Model. Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops. pp. 132-137 IEEE Computer Society (2008).
81. Letier, E., Lamsweerde, A.V.: Reasoning about partial goal satisfaction for requirements and design engineering. Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering. pp. 53-62 ACM, Newport Beach, CA, USA (2004).
82. University of Toronto: GRL - Goal-oriented Requirement Language, <http://www.cs.toronto.edu/km/GRL/>.
83. Dipartimento di Ingegneria e Scienza dell'Informazione - Università degli Studi di Trento: Tropos |, <http://troposproject.org/>.
84. University of Toronto: OME3 Home Page, <http://www.cs.toronto.edu/km/ome/>.
85. Maiden, N., Pavan, P., Gizikis, A., Clause, O., Kim, H., Zhu, X.: Making Decisions with Requirements: integrating i\* goal modelling and the AHP. Proceedings of the REFSQ'2002 workshop. , Essen, Germany (2002).
86. Sawyer, P., Bencomo, N., Hughes, D., Grace, P., Goldsby, H.J., Cheng, B.H.C.: Visualizing the Analysis of Dynamically Adaptive Systems Using i\* and DSLs. Proceedings of the Second International Workshop on Requirements Engineering

- Visualization. p. 3 IEEE Computer Society (2007).
87. Welsh, K., Sawyer, P.: Requirements Tracing to Support Change in Dynamically Adaptive Systems. Proceedings of the 15th International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 59-73 Springer-Verlag, Amsterdam, The Netherlands (2009).
  88. Welsh, K., Sawyer, P.: Understanding the Scope of Uncertainty in Dynamically Adaptive Systems. Requirements Engineering: Foundation for Software Quality. pp. 2-16 (2010).
  89. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-functional requirements in software engineering. Springer (2000).
  90. Maiden, N., Manning, S., Jones, S., Greenwood, J.: Generating requirements from systems models using patterns: a case study. *Requir. Eng.* 10, 276-288 (2005).
  91. Santander, V.F.A., Castro, J.: Deriving Use Cases from Organizational Modeling. Proceedings of the 10th Anniversary IEEE Joint International Conference on Requirements Engineering. pp. 32-42 IEEE Computer Society (2002).
  92. Feather, M.S., Fickas, S., Lamsweerde, A.V., Ponsard, C.: Reconciling System Requirements and Runtime Behavior. Proceedings of the 9th international workshop on Software specification and design. p. 50 IEEE Computer Society (1998).
  93. Robinson, W.N.: Monitoring Web Service Requirements. Requirements Engineering, IEEE International Conference on. p. 65 IEEE Computer Society, Los Alamitos, CA, USA (2003).
  94. Lamsweerde, A.V., Letier, E.: Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Trans. Softw. Eng.* 26, 978-1005 (2000).
  95. Robinson, W.: Monitoring Software Requirements Using Instrumented Code. Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9 - Volume 9. p. 276.2 IEEE Computer Society (2002).
  96. Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E.: Requirements Reflection: Requirements as Runtime Entities. Track on New Ideas and Emerging Results (NIER). , Cape Town, South Africa (2010).
  97. Sawyer, P., Bencomo, N., Whittle, J., Letier, E., Finkelstein, A.: Requirements-Aware

- Systems A research agenda for RE for self-adaptive systems. , Sydney, Australia (2010).
98. Amyot, D., Mussbacher, G.: URN: Towards a New Standard for the Visual Description of Requirements. Proceedings of the Third International Workshop on Telecommunications and Beyond: the Broader Applicability of SDL and MSC. 24--26 (2002).
  99. The Xiph Open Source community: Theora.org :: main - Theora, video for everyone, <http://www.theora.org/>.
  100. Flash, iPhone, Silverlight media server and more - Wowza Media Server, <http://www.wowzamedia.com/>.
  101. Lapouchnian, A., Mylopoulos, J.: Modeling Domain Variability in Requirements Engineering with Contexts. Conceptual Modeling - ER 2009. pp. 115-130 (2009).
  102. University of Toronto: i\* Wiki : Guideline (Beginner,Concept) To indicate that a Goal can be achieved by performing several sub-tasks, model the decomposition by introducing a Task., [http://istar.rwth-aachen.de/tiki-index.php?page\\_ref\\_id=206](http://istar.rwth-aachen.de/tiki-index.php?page_ref_id=206).
  103. Ramesh, B., Jarke, M.: Toward Reference Models for Requirements Traceability. IEEE Trans. Softw. Eng. 27, 58-93 (2001).
  104. Xavier Franch, Gemma Grau, Enric Mayol, Carme Quer, Claudia Ayala, Carlos Cares, Fredy Navarrete, Mariela Haya, Pere Botella: Systematic Construction of i\* Strategic Dependency Models for Socio-Technical Systems, <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.142.1436>, (2007).
  105. Lucena, M., Castro, J., Silva, C., Alencar, F., Santos, E., Pimentel, J.: A Model Transformation Approach to Derive Architectural Models from Goal-Oriented Requirements Models. Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: ADI, CAMS, EI2N, ISDE, IWSSA, MONET, OnToContent, ODIS, ORM, OTM Academy, SWWS, SEMELS, Beyond SAWSDL, and COMBEK 2009. pp. 370-380 Springer-Verlag, Vilamoura, Portugal (2009).
  106. Alencar, F., Moreira, A., Araújo, J., Castro, J., Silva, C., Mylopoulos, J.: Using Aspects to Simplify i\* Models. Proceedings of the 14th IEEE International Requirements

- Engineering Conference. pp. 328-329 IEEE Computer Society (2006).
107. Bencomo, N., Grace, P., Flores, C., Hughes, D., Blair, G.: Genie: supporting the model driven development of reflective, component-based adaptive systems. Proceedings of the 30th international conference on Software engineering. pp. 811-814 ACM, Leipzig, Germany (2008).
  108. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. Proceedings of the 8th International Conference on Advances Information System Engineering. pp. 1-21 Springer-Verlag (1996).
  109. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. (1990).
  110. Bencomo, N., Sawyer, P., Blair, G.S., Grace, P.: Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. SPLC (2). pp. 23-32 (2008).
  111. Welsh, K., Sawyer, P.: Deriving Adaptive Behaviour from i\* Models. , Hammamet, Tunisia (2010).
  112. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. IEEE Computer. 42, 22-27 (2009).
  113. Grace, P., Hughes, D., Porter, B., Coulson, G., Blair, G.: Middleware support for dynamic reconfiguration in sensor networks. (2007).