Requirements Tracing to Support Change in Dynamically Adaptive Systems

Kristopher Welsh and Pete Sawyer

Lancaster University, Computing Dept., Infolab21 LA1 4WA Lancaster, UK
{k.welsh,p.sawyer}@lancs.ac.uk

Abstract. [Context and motivation] All systems are susceptible to the need for change, with the desire to operate in changeable environments driving the need for software adaptation. A Dynamically Adaptive System (DAS) adjusts its behaviour autonomously at runtime in order to accommodate changes in its operating environment, which are anticipated in the system's requirements specification. **[Question/Problem]** In this paper, we argue that Dynamic Adaptive Systems' requirements specifications are more susceptible to change than those of traditional static systems. We propose an extension to i* strategic rationale models to aid in changing a DAS. **[Principal Ideas/Results]** By selecting some of the types of tracing proposed for the most complex systems and supporting them for DAS modelling, it becomes possible to handle change to a DAS' requirements efficiently, whilst still allowing artefacts to be stored in a Requirements Management tool to mitigate additional complexity. **[Contribution]** The paper identifies different classes of change that a DAS' requirements may be subjected to, and illustrates with a case study how additional tracing information can support the making of each class of change.

Keywords: Adaptive Systems, Requirements Evolution, Traceability.

1 Introduction

Changes can be required of a system at any stage during its design, implementation or useful life. In traditional *static* systems, these adaptations (e.g. changed requirements), are made offline by the system developers as maintenance. Recently a new class of system has begun to emerge, capable of adapting to changes in its environment autonomously at run-time. Such *Self-Adaptive* or *Dynamically Adaptive Systems* (DASs) are designed for volatile environments where the system requirements and/or their priorities may change along with the environment even while the system is running. The nature of the problem domains for which DASs are conceived are such that their environments may be only partially understood at design time. Similarly and particularly for embedded DASs, the potential for new and exploitable technologies to emerge during the course of the system's life is high because, with the current state-of-the-art, a DAS often represents a novel application of emergent technologies. The DAS itself 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.

M. Glinz and P. Heymans (Eds.): REFSQ 2009, LNCS 5512, pp. 59–73, 2009.

Thus, far from their runtime adaptive capability making them immune to the need for offline adaptation, DASs are particularly susceptible to it.

It has long been recognized [1] that requirements management (RM) is needed to help deal with anticipated change by recording (among other items of information) as traces the relationships between requirements and the down-stream artifacts of the development process. This allows questions about how requirements came to be and how decisions were reached to be answered later in the development process, or indeed after deployment. Given the identified need for evolvability and the variety of factors that may mandate it, the importance of traceability information in a DAS is, we claim, at least as high or even higher than in a comparable static system.

In this paper we identify requirements for traceability of DASs and, building on our earlier work on using goal models to discover DAS requirements [2], [3] show how i* [12] models can be augmented to record this information, for later integration in a Requirements Management tool.

The rest of this paper is organised as follows: Section 2 looks at related work in the area of DASs. Section 3 identifies the types of change that the specification of a DAS may need to accommodate, section 4 examines the types of traceability required to support these changes. Section 5 proposes a lightweight, i* based method of capturing these types of traceability information for the purposes of modelling the proposed DAS' adaptive behaviour. Section 6 presents a case study demonstrating the use of Section 5's method to revisit decisions in light of different types of change, whilst Section 7 concludes the paper.

2 Related Work

The requirements engineering (RE) community has recently started to investigate higher-level runtime representations that would support self-adaptation. Although the challenges posed by DASs to RE were first identified over ten years ago – principally, in run-time monitoring of requirements conformance [4] [5] [6] [7] – there are few current approaches for reasoning at runtime about system requirements.

There is a strong case that any such approach should be goal-based and a number of authors [8], [9], [10], [3] report on the use of goals for modelling requirements for DASs. This work commonly recognises that the aim of the system should be to *satisfice* its overall goals even as the environment in which it operates changes. Here, adaptation is seen as the means to maintain goal satisficement, while goal modelling notations such as KAOS [11] and i* [12] support reasoning about both functional and non-functional (*soft*-) goals. We have previously argued [2] that context-dependent variation in the acceptable trade-offs between non-functional requirements is a key indicator of problems that require dynamically adaptive solutions.

Goldsby *et. al*. [3] use i* as part of the LoREM process which partially implements the four levels of RE for self-adaptive systems proposed by Berry *et al.* [13]. The latter work is interesting because it partitions the environment into discrete *domains*, each of which represent a state of the environment with distinct requirements. The system's adapted configuration for each domain is termed a *target system*. LoREM has validated this approach on the requirements for a self-adaptive flood warning system implemented in the GridKit adaptive middleware system [14] which we use in

this paper as a case study to illustrate our approach to tracing. To date, DASs have not attracted any special attention that we are aware of from the RE research community for the challenges they pose to RM and tracing.

Ramesh and Jarke [15] categorise users of traceability information into two groups: *high* and *low*-level, depending on the types of use made of the information. Two traceability uses are of particular interest to DASs: traceability of decision rationale and for evolvability. To allow decision rationale tracing, a record needs to be made of all the viable alternatives considered, along with assumptions of the impact the selection of each alternative would have. Typically, this sort of information is only kept by so-called high-level traceability users, working on large, complex projects; with patchy coverage of rejected alternatives. Requirements evolution can occur due to a change in user needs, or due to an identified deficiency in the system. As requirements at different levels of detail change, the question "Where did this requirement come from?" becomes harder to answer. By explicitly recording changes to requirements, it becomes possible to understand how the system has evolved (or needs to) over time, and to identify derived requirements that need to be revisited in the light of new changes. Given the increased likelihood of change impacting a DAS' requirements, the likelihood of encountering derived requirements whose necessity is hard to establish, or even understand is also, we argue, increased.

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. The likelihood of far-reaching change impact means that traceability of decision rationale and for evolvability are crucially important for DAS developers as a consequence. Therefore, we argue that DASs promote high-level traceability as essential practice.

3 Types of Change

Although a DAS can adjust its behaviour in response to change whilst in operation, this does not mean that all changes can be handled automatically by the system. New requirements, new technology or simply a better understanding of the environment may all require the system to be re-specified, in whole or in part. If already deployed, the system will need to be taken offline for static adaptation to be carried out. The static adaptation process is not radically different to that of a traditional (non-adaptive) system, but the relative complexity of a DAS coupled with the increased likelihood of change means that an inefficient change management process will rapidly become problematic.

Our work [2] [16] builds upon Berry *et. al.*'s four levels of RE for Dynamic Adaptive Systems [13]. Berry *et al* start with the assumption that the environment in which a DAS must operate can be characterized as discrete states or *domains*. Each domain is served by a separate *target system*, which in a DAS is a conceptualization of a system configuration. Hence, when the environment makes the transition for domain 1 to domain 2, the DAS adapts from target system S1 to target system S2.

We use i* [12] to model the system at each level. Level 1 RE is done on a per-target system basis, specifying how the system functions in each. Level 2 RE specifes

which target system to adopt given the runtime context. Level 3 RE specifies the requirements on the system's adaptation mechanism, and how it achieves its level 2 RE. Level 4 RE is the most abstract level, covering adaptation mechanisms in general. Most changes will involve some modification to the level 1 models, which is the focus of this work. In cases where a previously made decision is changed, the system's level 2 and 3 models may also be affected; these secondary impacts are the subject of ongoing work.

We have identified five distinct classes of change that a DAS' specification may need to be adjusted for. Each needs handling differently.

- **Environmental Change.** This will be particularly common given the inherent volatility of a DAS' 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 system's life cycle, and the need for change may be punctuated with a system failure if it emerges only after deployment. A change to a previously identified environmental domain will trigger the need to re-evaluate the design decisions taken for it, whereas 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.
- **Broken Assumption.** This may be an assumption about the environment in a given domain ("There will be ample solar power during the day"), or an assumption about a given system component's suitability for a domain ("Communicating via Bluetooth will use less power than Wi-fi"). The assumptions underpinning the level 2 modelling ("The environment will not switch directly from S1 to S6" or "The system will spend most of its time in S1") 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 system, or may only become apparent after deployment. An assumption being broken triggers the need to re-evaluate all decisions based upon it.
- **New Technology.** The availability of a new technology that can be exploited can be modelled as a new alternative for a decision. Given the relative immaturity of adaptation frameworks this will likely occur frequently. As with a static 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 designers will need to make the decision for each target system. If the new technology is utilised in one or more targets, other decisions that impact on the same quality features as the new technology in these targets will need to be revisited.
- **Consequential Change.** This is 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.

- **User Requirements Change.** This class of 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 impact.

4 Traceability Requirements

Given the increased likelihood of change impacting the requirements specification of a DAS, and the fact that decisions are essentially taken again and again for each target system, traceability information becomes more important. Each of the types of change discussed in the previous section has differing traceability requirements.

- **Environmental Change.** When environmental understanding changes, an entire model 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 in light of the changed (or new) environmental knowledge. For this type of change, the only traceability information needed is the ability to identify all the decisions taken for a given target system. In most cases, the target systems of a DAS will share much commonality, perhaps exhibiting variability only in a small number of discrete components. As such, each's level 1 model will merely be a refinement of another's.
- **Broken Assumption.** When an assumption has been broken, either by improved understanding of the environment or available alternatives, or having been broken demonstrably in the field, all of the decisions based on this assumption will need to be revisited. In order to facilitate this forward tracing, there needs to be a record of all decisions reliant on this assumption, along with the information on alternatives required to re-make them.
- **New Technology.** A new decision alternative, often brought about as a result of newly available technology, will require only a limited number of decisions to be revisited. However, each decision may need to be revisited for each target system. As such, to support this backwards tracing, there needs to be a record of alternatives previously considered and what basis the previous decision was taken on.
- **Consequential Change.** A consequential change, so named because the change is necessitated as a consequence of a previous 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.
- **User Requirements Change.** A user requirement change can potentially affect any or all target systems, or may necessitate a change to a static (i.e. identical in all targets) component of the system. Therefore, as with user requirement changes to static systems, the impact of this type of change varies from case to case.

From the range of traceability requirements above we can conclude that for decisions to be re-visited and re-evaluated it is necessary to record all of the alternatives considered, along with the rationale behind the selection made. Each alternative requires assumptions that impact on the system's competing requirements to be recorded, along with the presumed impact itself. The relative importance of the system's competing requirements in this domain drives per-target selection decisions, and should be recorded along with the environmental assumptions that underpin the re-prioritisation.

5 Recording Traceability Information

In our earlier work [3] we argue that DAS development should commence with an early goal modelling phase that allows the analyst to reason about the environment, the goals that the actors within that environment have and the softgoals that constrain the quality with which the goals can be satisfied. We use the i* modelling language [12] in our LoREM [16] method for developing the requirements for DASs since i* is well matched to these aims.

There are two types of i* model: the Strategic Dependency (SD) and Strategic Rationale (SR) models. The SD model is intended to identify the goals and softgoals in complex, multi-agent systems and the dependencies between agents engaged in attempting to satisfy the system's goals and softgoals. The SR model is intended to explore the available alternatives for satisfying each agent's goals and softgoals. The SR models offer a useful means of reasoning about trade-offs, primarily in terms of the extent to which the various solution alternatives satisfy the softgoals. As the "rationale" in SR suggests, SR models serve to record the rationale for decisions. Hence, i* can be thought of as not only a modelling technique, but also as a means for tracing the motivation for requirements. It is this feature that we propose to exploit here.

However, although an i* SR model allows us to infer an actor's basis for making a decision by examining the decision's impact on system goals, the conveyed information on impacts is limited, and understanding complex decisions that balance conflicting goals, such as most adaptation decisions, is difficult. The NFR Framework [17] to which i* is related includes a mechanism for recording beliefs and assumptions, referred to as *claims*. A claim can be used to support a decision in two ways: attached to a node on an NFR model, it represents a comment about an alternative, which may influence its selection; attached to a link, it represents a comment on the importance (not the magnitude) of the impact. By adding claims to i* SR diagrams, it is possible to convey similar information, allowing decision rationale to be inferred more effectively.

A claim may be used to explain a decision alternative's negative impact on the system softgoal "minimise running costs" by claiming that this alternative "requires an additional operator for monitoring". On a standard SR diagram, by contrast, only the negative impact itself would be recorded, with the rationale merely implicit and thus potentially inexplicable to a developer tasked with evolving the system. A claim could also be used to "de-prioritise" a high magnitude impact. For example encrypting an intranet site may significantly hinder scalability, but still be deemed necessary for security, despite the positive impact on the security goal being weaker.

In most instances, it is possible to record a decision justification using a claim in two ways. The first is to invalidate a selected candidate's negative impact(s) by claiming they are insignificant or unavoidable, or to promote an alternative's claiming them too costly. The second is to promote a candidate's positive impact(s) by claiming them necessary, or to invalidate an alternative's claiming them needless. Although both are equivalent when revisiting decisions later, and those with scalability concerns may wish to record the rationale using the fewest additional entities, we prefer to record whichever way is closest to the rationale actually used. Figure 1 shows an example of an NFR claim being used to justify a decision using the first method: promoting the rejected alternative's negative impacts. The selected alternative is coloured white for clarity.

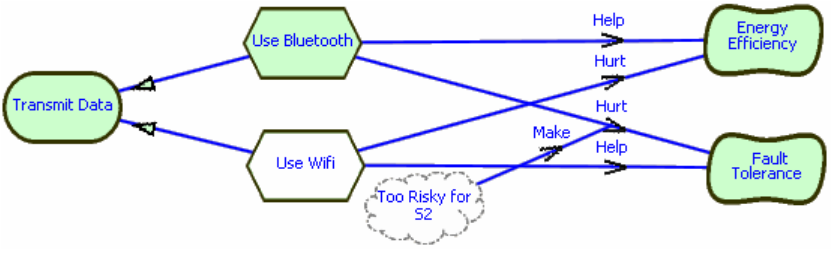

Fig. 1. NFR claim justifying selection of a component negatively impacting Energy Efficiency

In this example, the DAS under analysis is a wireless sensor network and one of its goals is to "Transmit data". This has spurred the need to select a wireless communication standard and two alternatives have been identified, modelled as: *Use Bluetooth* and *Use WiFi* (IEEE 802.11). When alternatives such as these exist, reasoning about which one to select is aided by considering their impact on softgoals. In Figure 1, two softgoals have to be satisfied: "Energy efficiency" and "Fault tolerance". These two softgoals are imperfectly compatible, thus spurring the kind of high-level trade-off analysis that i* is so useful for.

The relatively short-range Bluetooth solution is energy efficient but its ability to tolerate a node failure is constrained: since if data is routed via nodes A and B, and node B fails, the next closest node may be too remote to establish a connection with A. The WiFi alternative, by contrast, endows the nodes with greater communication range at the cost of (relatively) high power consumption. These relative strengths and weaknesses are captured crudely by the "helps" and "hurts" *contribution links* between the alternative solutions and the softgoals. It is these contribution links that we propose should be annotated with claims.

As the environment cycles through its set of possible domains, the DAS needs to adapt to the appropriate target system. Analysis of the environment shows that much of the time, the DAS will operate in a benign domain in which the risk of node failure is low. However, another domain exists, for which the target system is labeled "S2", where the risk of node-failure is significant. A claim attached to the contribution link between the "Use Bluetooth" task and the "Fault tolerance" softgoal records the rationale for why the extent to which selecting Bluetooth as the communication standard hurts fault tolerance makes its selection unacceptable for S2.

Note that this is different from using fine-grained contribution links (*Make*, *Break*, *Some*+, ++, etc.) because although defining impact magnitude, the weight (importance) of the contribution link is context- (i.e. domain/target system) dependent. Nor are claims the same as i* "beliefs" which represent assumptions made by an actor, with no presumption of truthfulness by the analyst. In the next section we expand upon the wireless sensor network example.

6 Case Study

GridStix [19] is a DAS built to perform flood monitoring and prediction, and is deployed on the River Ribble in North West England. 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 nodes have processing capability and memory. This allows processing of the data and execution of the predictive models of the river's behaviour to be performed on site with the system acting as a lightweight grid. However, the nodes are resource-constrained and some tasks are best performed by distributing computation among the nodes. Distributing computation has a cost in terms of the power consumed by inter-node communication, however. This is a serious issue since GridStix's location is remote and power has to be provided by batteries and solar panels, which provide only limited power.

Domain experts identified three distinct environmental domains that GridStix needs to operate in. In the "quiescent" domain, the river has a low depth and flows relatively slowly. In the "high flow" domain, the river flows faster, but is still at a relatively low depth, this can presage the rapid onset of the third domain "flood", where the river depth has started to increase, and there is imminent danger of flooding, which poses a danger to both local residents and the system.

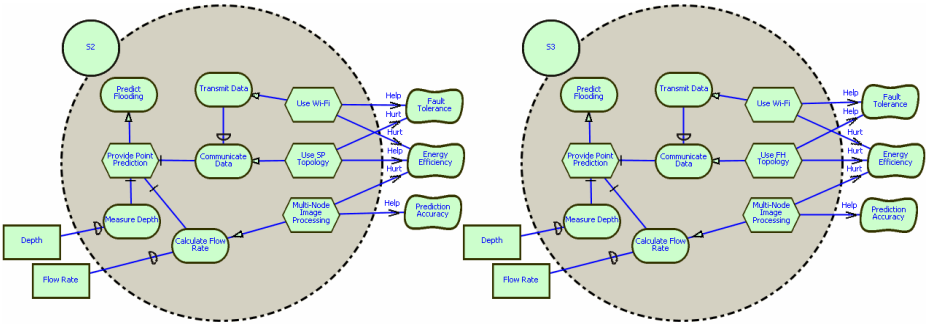

Fig. 2. Models of GridStix configured for *High Flow* (S2) and *Flood* (S3) domains

GridStix's key softgoals are "Energy efficiency" to maximise battery life, "Prediction Accuracy" to provide timely and accurate flood warnings, and "Fault Tolerance" for survivability. The system is built upon the Gridkit middleware [14] which allows software components to be substituted at runtime. Our practice is to develop a separate SR diagram for each target system. Figure 2 shows part of those for target system

S2, which operates in the high flow domain, and S3 which operates in the flood domain. The actual, deployed system varies a wider set of individual components dynamically, but their inclusion would increase the models' complexity for little or no illustrative benefit.

The S2 and S3 SR diagrams illustrate which decisions were reached for each target system. Notice that the only difference between S2 and S3 is that in S3, the task "Use FH topology" substitutes for "Use SP topology" in S2. SP and FH represent spanning tree algorithms used to communicate data among the nodes in the network: shortest path and fewest hop, respectively. The characteristics of the spanning tree algorithms are different and this is reflected by their respective contribution links to the Fault tolerance and Energy efficiency softgoals.

From the contribution links from the two spanning tree tasks in S2 and S3, it is possible to infer that the "Energy efficiency" softgoal was de-prioritised between S2 and S3. It is not possible, however, to revisit the decisions in light of new information or some other change, given that the alternatives considered are not documented, and that the only justification for the decision recorded is that: "This alternative helps the Prediction Accuracy softgoal, which must have been prized more highly than Energy Efficiency." The real rationale for the choice of spanning tree algorithm has been lost, *viz* in the h*igh flow* domain the risk of immersion or water-borne debris destroying a node is slight, so the need to conserve energy in case the situation worsens takes priority. The relatively energy-efficient shortest path algorithm is therefore the better choice. In the *flood* domain, however, node damage is a real risk and the relatively power-hungry but resilient fewest hop algorithm is used to favour *Fault tolerance* over *Energy efficiency*.

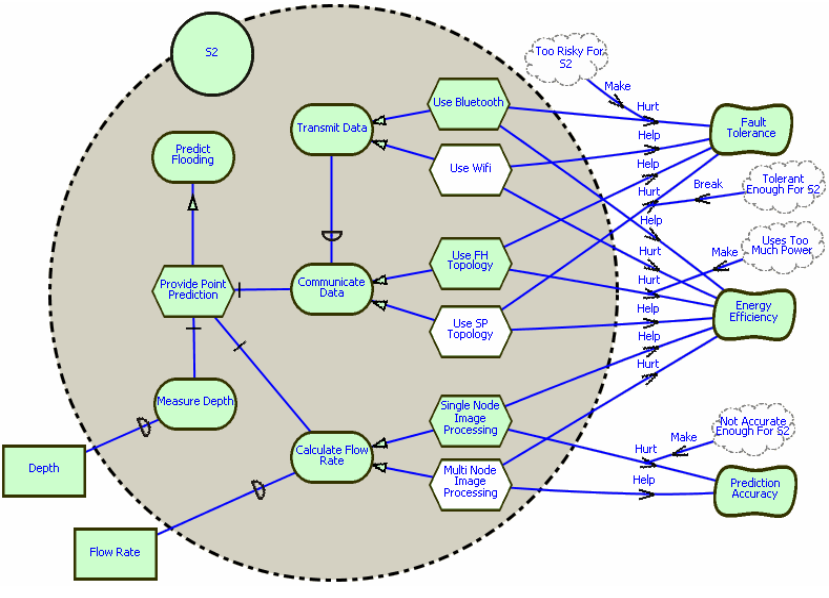

Fig. 3. Augmented model of GridStix in the *High Flow* (S2) domain

Although the limited traceability available from Figure 2 is far better than nothing, dealing with numerous, frequent specification changes and having to adjust several target's specifications using only this information is a bleak prospect. Recording the rejected alternatives and a small amount of additional information using NFR claims could make these changes far more manageable. Figures 3 and 4 show the same two models, with this extra information recorded.

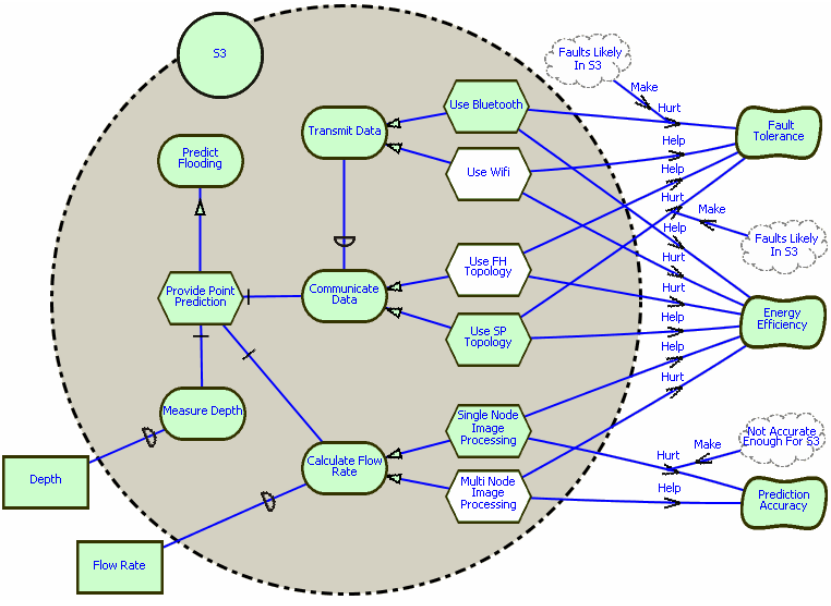

Fig. 4. Augmented model of GridStix in the *Flood* (S3) domain

The additional information shown in Figures 3 & 4 allows rejected alternatives to be re-examined if necessary with a richer (although still incomplete) understanding of the previous decision's basis. We believe that the information recorded in Figures 3 & 4 is the minimum required to allow per-target system decisions to be revisited.

In section 3, we identified five distinct classes of change that a DAS' specification may need to be adjusted for. We now discuss how recording additional traceability information allows some of these types of changes to be carried out more easily.

Environmental Change. The first type of change identified was Environmental change. This could take the form of a new environmental constraint, an adjustment to domain boundaries, or an entire domain being introduced, eliminated, or merged with another. We illustrate the new domain scenario, by introducing a fourth domain to the GridStix system. This fourth domain, blockage occurs when a natural or man-made obstruction hinders the flow of the river downstream from the GridStix nodes.

Although this fourth domain does not necessarily pose a danger to local residents and should not trigger a flood alert, there is a significant risk of nodes becoming submerged and quickly failing. This makes the more power efficient shortest path (SP) networking algorithm and Bluetooth communication risky choices. Furthermore, if the blockage is removed (or breached) suddenly, the river's behaviour could be very

difficult to predict without data downstream of the blockage. Therefore, it would be desirable for the system to report this condition to the Environment Agency, who would typically seek to remove occluding objects maintaining the river's regular flow. The blockage domain is characterised by a high river depth and low flow rate. It could be argued that the new domain and the requirement to report it are two separate changes (one environmental change and one user requirement change), but we have modelled both together for brevity. Figure 5 shows the SR model for the target system, S4, for the blockage domain.

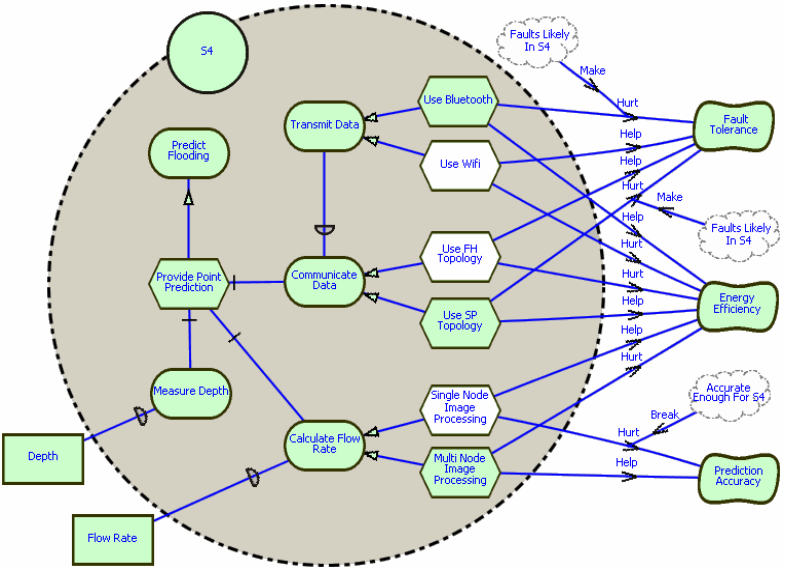

Fig. 5. Augmented model of GridStix configured for the *blockage* (S4) domain

The decisions taken for the Blockage (S4) domain closely mirror those taken for the Flood (S3) domain, with both Bluetooth and Shortest Path networking rejected on the grounds that they each compromise fault tolerance unacceptably, and that there is a real risk of node failure in this domain. Unlike the Flood (S3) domain, the chances of a sudden increase in flow rate or depth are remote, and the slower, more power efficient single node image processing algorithm can be used to save power.

Broken Assumption. The second class of change identified was *broken assumption*. This can happen as understanding of the operating environment improves, or as knowledge of available components becomes more complete. Dealing with this type of change involves tracing all the decisions made based on the assumption, and revisiting them.

To illustrate this type of change, we have modified our model of the Flood (S3) domain, replacing the assumption "Single node image processing is not accurate enough for S3" with the inverse. In this instance, changes are confined to this model, although this will not always be the case. In fact, only one decision is reached on the basis of this assumption, and Figure 6 shows single node image processing being used instead, along with the replaced assumption.

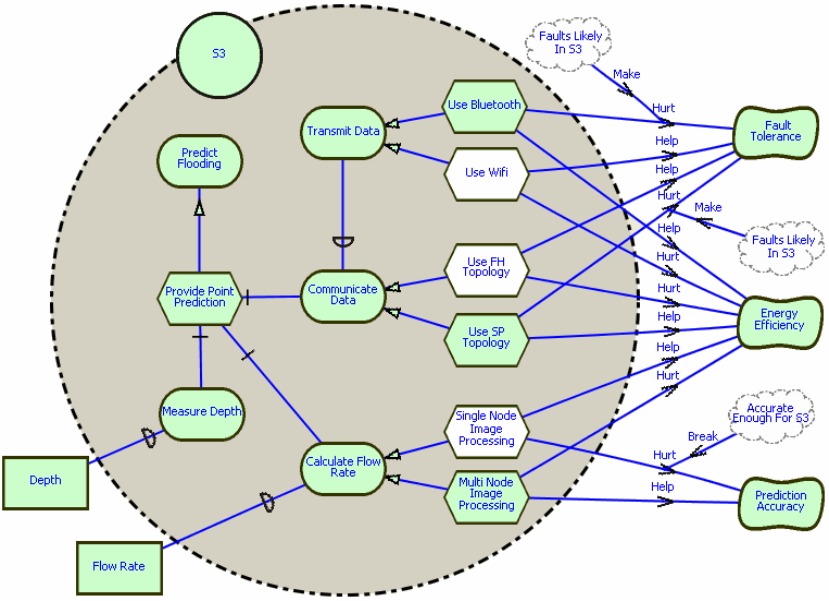

Fig. 6. Augmented model of GridStix in the *Flood* (S3) domain after a broken assumption

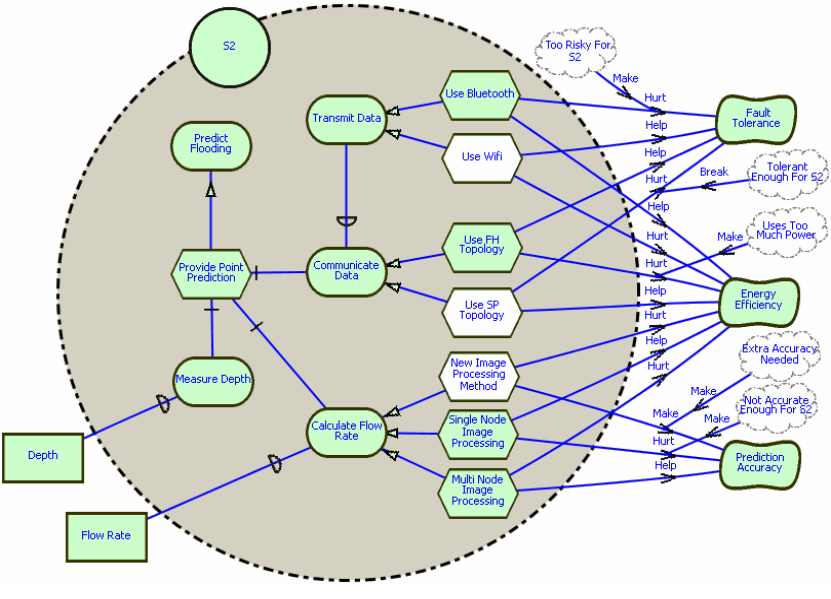

Fig 7. Augmented model of GridStix in *High Flow* (S2) domain, with new decision alternative

This class of change is particularly amenable to analysis in a requirements management tool, providing the tool has a record of decisions reliant on the now false assumption. By automating the location of affected decisions, the overhead of dealing with this class of change will be decreased, particularly in larger systems.

New Technology. The third class of change identified was a *new technology* that becomes available for use. This may introduce a brand new decision to be taken for each domain, or add a new alternative to a decision already taken. It is this second variant that we have chosen to illustrate in Figure 7, which shows the prediction model decision after being re-taken with a newly available alternative.

Although Figure 7 shows just the *High Flow* domain, the new alternative would need to be added to each diagram, creating a significant overhead in systems with many domains. The new/changed decision in each needs to be re-examined and remade in light of the new alternative, which may be selected in some, all or none of the target systems. In this example and domain, the extra accuracy offered by the new image processing method was deemed necessary and its negative impact on energy efficiency tolerated on the basis of this claimed need.

7 Conclusion

In this paper we have argued that, far from removing the need for off-line adaptation, DASs are inherently susceptible to the need to adapt in ways that are beyond the scope of their (necessarily limited) self-adaptive capabilities, which are limited in scope at design time. In practical terms, to give a DAS usefully long service life, it is likely to need to be maintained by human developers to correct defects and to take advantage of new knowledge and new capabilities. As such, a DAS' requirements specification needs to be as amenable to change as the system itself.

We have classified some of the types of change that a DAS is subject to and argued that some are special to DASs. From this analysis we have identified a need to trace how these change types impact on the requirements. To evaluate this, we have applied three of the types of identified change (Environmental Change, Broken Assumption and New Technology) to a case study we have used in our earlier work [2], [3], [16]. In this earlier work we developed a process for analysing DASs based on goal modelling using i*. We have therefore proposed recording rejected decision alternatives alongside the accepted option in i* Strategic Rationale models, which would previously have shown only the selected alternative. We have also proposed extending i* with the notion of *claims* which we have borrowed from the related NFR framework.

Claims permit us to annotate the contribution links used in i* Strategic Rationale models with the rationale underpinning a decision, explaining how it was reached in terms of the softgoals affected. The annotation also makes explicit the re-prioritisation of softgoals between domains, which previously had to be inferred by comparing several target system's models and examining differences in contribution links.

Ultimately, we envisage a target system's decision alternatives, their presumed impact on system softgoals, the selected option and the rationale underpinning the selection decision itself (recorded as claims) being mapped into a conventional tracing tool suck as DOORS [18]. Such a tool would allow the now-possible tracing to be automated, bringing greater benefit in terms of efficiency with scale.

References

1. Gotel, O., Finkelstein, A.: An analysis of the requirements traceability problem. In: Proceedings of the International Conference on Requirements Engineering, Colorado Springs (1994)
2. Welsh, K., Sawyer, P.: When to Adapt? Identification of Problem Domains for Adaptive Systems. In: Paech, B., Rolland, C. (eds.) REFSQ 2008. LNCS, vol. 5025, pp. 198–203. Springer, Heidelberg (2008)
3. Goldsby, J., Sawyer, P., Bencomo, N., Cheng, B., Hughes, D.: Goal-Based Modelling of Dynamically Adaptive System Requirements. In: Proceedings of 15th IEEE International Conference on Engineering of Computer-Based Systems, Belfast, Northern Ireland (2008)
4. Fickas, S., Feather, S.: Requirements Monitoring in Dynamic Environments. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering, York, England (1995)
5. Savor, T., Seviora, R.: An approach to automatic detection of software failures in realtime systems. In: IEEE Real- Time Tech. and Appl. Sym., pp. 136–147 (1997)
6. Feather, M., Fickas, S., van Lamsweerde, A., Ponsard, C.: Reconciling system requirements and runtime behavior. In: Proceedings of the 9th International Workshop on Software Specification and Design (1998)
7. Robinson, W.: A requirements monitoring framework for enterprise systems. Requirements Engineering 11(1), 17–41 (2006)
8. Yu, Y., Leite, J., Mylopoulos, J.: From goals to aspects: Discovering aspects from requirements goal models. In: Proceedings of the 12th IEEE International Conference on Requirements Engineering (2004)
9. Lapouchnian, A., Liaskos, S., Mylopoulos, J., Yu, Y.: Towards requirements-driven autonomic systems design. In: Proceedings of 2005 Workshop on Design and Evolution of Autonomic Application Software, St. Louis, Missouri, USA (2005)
10. Yu, Y., Mylopoulos, J., Lapouchnian, A., Liaskos, S., Leite, J.: From stakeholder goals to high-variability software design. Technical report csrg-509, University of Toronto (2005)
11. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. 20, 3–50 (1993)
12. Yu, E.: Towards modelling and reasoning support for early-phase requirements engineering. Requirements Engineering. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering (1997)
13. Berry, D., Cheng, B., Zhang, J.: The four levels of requirements engineering for and in dynamic adaptive systems. In: Proceedings of the 11th International Workshop on Requirements Engineering: Foundation for Software Quality (2005)
14. Coulson, G., Grace, P., Blair, G., Cai, W., Cooper, C., Duce, D., Mathy, L., Yeung, W., Porter, B., Sagar, M., Li, W.: A component-based middleware framework for configurable and reconfigurable Grid computing: Research Articles. Concurr. Comput. Pract. Exper. 18(8), 865–874 (2006)

15. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Transactions on Software Engineering 27(1), 58–93 (2001)
16. Sawyer, P., Bencomo, N., Hughes, D., Grace, P., Goldsby, H.J., Cheng, B.H.: Visualizing the Analysis of Dynamically Adaptive Systems Using i* and DSLs. In: Proceedings of the Second international Workshop on Requirements Engineering Visualization (2007)
17. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional requirements in software engineering. Kluwer Academic Publishers, Dordrecht (2000)
18. Quality Systems & Software Ltd., Oxford Science Park, Oxford, U.K., DOORS Reference Manual (V3. 0) (1996)
19. Hughes, D., Greenwood, P., Coulson, G., Blair, G.: GridStix: supporting flood prediction using embedded hardware and next generation grid middleware. World of Wireless, Mobile and Multimedia Networks (2006)