

Visualizing the Analysis of Dynamically Adaptive Systems Using *i** and DSLs*

Pete Sawyer¹, Nelly Bencomo¹, Danny Hughes¹, Paul Grace², Heather J. Goldsby³, Betty H. C. Cheng³

¹Computing department, InfoLab21, Lancaster University, LA1 4WA, United Kingdom

²Departement Computerwetenschappen, Katholieke Universiteit Leuven, B-3001 Heverlee, Belgium

³Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824, USA

email: {sawyer,nelly,danny}@comp.lancs.ac.uk;{Paul.Grace}@cs.kuleuven.be {hjg,chengb}@cse.msu.edu

Abstract

*Self-adaptation is emerging as a crucial enabling capability for many applications, particularly those deployed in dynamically changing environments. One key challenge posed by Dynamically Adaptive Systems (DASs) is the need to handle changes to the requirements and corresponding behavior of a DAS in response to varying environmental conditions. In this paper we propose a visual model-driven approach that uses the *i** modeling language to represent goal models for the DAS requirements. Our approach applies a rigorous separation of concerns between the requirements for the DAS to operate in stable conditions and those that enable it to adapt at run-time to enable it to cope with changes in its environment. We further show how requirements derived from the *i** modeling can be used by a domain-specific language to achieve requirements model-driven development. We describe our experiences with applying this approach to GridStix, an adaptive flood warning system, deployed on the River Ribble in North Yorkshire, England.*

1. Introduction

It is generally accepted that errors in requirements are very costly to fix [22]. The avoidance of erroneous requirements is particularly important for the emerging class of systems that need to adapt dynamically to changes in their environment. Many such Dynamically Adaptive Systems (DASs) are being conceived for applications that require a high degree of assurance [23, 27], in which an erroneous re-

quirement may result in a failure at run-time that has serious consequences. Of course, the requirement for high assurance is not unique to DASs, but the requirement for dynamic adaptation introduces complexity of a kind not seen in conventional systems where adaptation, if it is needed at all, can be done off-line. This dynamic adaptation-consequent complexity is manifested at all levels, from the services offered by the run-time platform, to the analytical tools needed to understand the environment in which the DAS must operate.

Recent years have seen much progress in designing platforms and programming frameworks that support DAS developers [15, 16, 18, 20, 24, 25]. Component frameworks and service architectures are emerging that permit architectural reconfiguration at run time to, for example, substitute networking technologies on-the-fly. While the technologies that enable systems to adapt are developing quickly, support for understanding the conditions in the problem domain that act as the stimuli for system adaptation, and for translating this understanding into requirements is lacking. We believe that useful DAS deployment will be inhibited until we are able to equip analysts with the tools to help them specify the requirements for the highly complex problems they need to solve.

The focus of our work is on closing the gap between understanding DAS requirements and implementing them. In this paper we describe how we combine the goal-oriented modeling of problems that demand DAS solutions with the model-driven development of DAS architectures.

Our approach is fundamentally visual since we use *i** [28] to model states of the environment, the conditions that signify transitions between states in the environment, and conditions that determine when system adaptation needs to occur. We show how goal models can be refined and translated into a visual domain-specific language (DSL) for architecting middleware component configurations. We illustrate the utility of our approach using a deployed prototype of a flood warning system called GridStix.

*This work has been supported in part by NSF grants EIA-0000433, EIA-0130724, CDA-9700732, CCR-9901017, Department of the Navy, Office of Naval Research under Grant No. N00014-01-1-0744, Eaton Corporation, Siemens Corporate Research, a grant from Michigan State University's Quality Fund and EPSRC project EP/C010345/1 The Divergent Grid

Here, the behaviour of the river determines how the GridStix sensor network needs to adapt in order to satisfy a range of requirements whose relative priorities change as the river changes state. The DSL models that we derive from the requirements translate into policy rules used by the GridKit [15, 18] component framework that implements the GridStix. This *requirements* model-driven approach provides a visualization of the requirements model and provides traceability through the requirements and design artifacts of the development process.

The rest of this paper is structured as follows. In Section 2 we introduce the collection of models that we propose and show how a separation of concerns is imposed by considering three different levels of requirements analysis. In Section 3, we use the GridStix flood warning system to illustrate the application of our approach. Section 4 briefly surveys related work that supports the analysis of DASs and Section 5 concludes by identifying the next steps needed in our research.

2. Levels of Analysis

To be tractable to our requirements analysis approach, it must be possible to anticipate the behaviour of the environment in which a DAS will operate. This caveat excludes a range of hostile environments in which a DAS must react to unpredictable events. Such problems are beyond the scope of this work. Nevertheless, many challenging problem domains can be tackled if it is possible to characterize the environment as a finite set of stable states subject to events that cause transitions between states. A DAS that operates in such an environment can be modeled as a collection of steady-state programs, that we call *target systems*, each of which correspond to, and operate within, a state of the environment [31].

2.1. Levels of DAS analysis

The range of problems where the domain environment can be modeled as a finite set of states maps well onto the work of Berry *et al.* [3] who identified four levels of analysis that need to be performed for a DAS. Each level represents the analysis of a particular concern of a DAS. Here, we only present three of the four levels - those that are directly related to the requirements for a specific DAS. Furthermore, these three levels impose a useful separation of concerns for DAS requirements.

The concerns are the behaviour of the set of target systems, the requirements for how the DAS adapts from one target system to another, and the requirements for the adaptive infrastructure. We make each concern the focus of a different model, or group of models, that we use to visualize the DAS requirements.

Level 1 is concerned with *monitoring*. The analyst must understand the requirements of each target system; the adapted configuration of the DAS for a particular stable state of the environment. For example, the requirements for a car in which the engine was running normally would represent the requirements for the target system responsible for maintaining normal operation of the car. If a fault occurred, the car would be subject to a different set of requirements and controlled by a different target system. The fault might be detected by monitoring the exhaust gasses. Level 1 is therefore analogous to the analysis needed for a conventional, non-adaptive system with the crucial difference that:

- a separate Level 1 model is needed for each stable state of the environment;
- each Level 1 model must specify the variables in the environment to monitor that represent the triggers for adaptation.

Level 2 is concerned with *decision making* and has no direct analogue in conventional systems. Level 2 helps the analyst focus on understanding the requirements for adaptation by defining *adaptation scenarios*. The analyst must identify the values of the monitored environment variables that cue transitions between stable states, specify the target systems that represent the start and end points of an adaptation and specify what form the adaptation takes. For example, detection of impurities in the exhaust gasses might result in a requirement for the car to adopt a *limp home* mode, causing the DAS controlling the car to adapt from the normal operation target system to a target system that restricts the fuel supply, limits engine speed and so on. Berry *et al.* characterized Level 2 analysis as something performed by the DAS itself at runtime. However, a DAS can only adapt if a human analyst has identified the triggers for adaptation from their analysis of the environment. In our interpretation of the four-level model, therefore, we consider Level 2 to be performed by a human analyst, albeit indirectly at development time and executed at runtime.

Level 3 analysis is concerned with *adaptation*; identifying the requirements on the adaptive infrastructure in order to enable the DAS to adapt. An adaptation infrastructure comprises a set of mechanisms that enable adaptation to occur. The adaptation infrastructure must include three key types of mechanisms: a *monitoring mechanism*, a *decision-making mechanism*, and an *adaptation mechanism*. A monitoring mechanism is hardware or software responsible for detecting adaptation conditions (triggers) at run time. A decision-making mechanism is responsible for selecting a target system to adapt to, based on input from the monitoring mechanisms at run time. An adaptive mechanism executes adaptive steps at run time, where an *adaptive step* is an adaptive action, e.g., swapping a component.

The adaptation infrastructure is a key element of the platform that provides the run-time environment and programming tools needed to support DASs. In most cases, good engineering practice combined with sheer complexity preclude DAS development without the support of such a platform. Although such platforms support the satisfaction of the DAS requirements, they also constrain their satisfaction according to the features they provide and the qualities with which they can deliver them.

2.2. Modeling the levels of analysis with i^*

The approach we propose uses i^* to identify the actors, goals and softgoals in the problem domain in which the DAS will operate. We call this the Stakeholder Goal Model. Next, it is necessary to identify discrete stable states within the problem environment. While our approach cannot provide the domain expertise needed to identify states of the environment, it does provide the means to visualize the target systems (i.e. adapted states of the DAS) that correspond to each state of the environment.

Once the Stakeholder Goal Model has been created and the states of the environment are known, the explicit and implicit intentionality of the principal agents of the DAS are explored by developing i^* models. A large class of DASs are designed as autonomous embedded systems with little interaction with other agents. Hence much of the analysis will focus on the DAS itself rather than on other agents within the domain. However, since we need to understand and specify the adaptive behaviour of the DAS, i^* models of the DAS must be developed for each of the three levels above.

First, the analyst constructs a Level 1 model to analyze the requirements of each target system and the environment variables to monitor. Hence, for the car example used above, a Level 1 model would be developed to specify how the goals of the car could be met when the engine is operating normally, another model would be developed for when a non-critical fault is detected, another for when a safety system fails, and so on. Requiring the analyst to model these individually makes explicit the discrete stable states of the environment for which a target system must be specified. A useful side-effect is that it helps partition the complexity of the environment model. This is particularly useful since models described using graphical notations like i^* can become so large as to be difficult to view on a single piece of paper or a computer screen.

Once the Level 1 models have been constructed to specify the behaviour and the monitoring conditions have been identified for each target system, the analyst can develop the Level 2 models. A Level 2 model is required for each valid transition between states of the environment. These correspond to adaptations between target systems in the DAS.

Again, by requiring each adaptation to be modeled separately, we force each adaptation to be made explicit while mitigating the potential for the i^* models to become unwieldy.

A single Level 3 model is developed to specify how the adaptation infrastructure needs to be configured to enable the adaptations specified in the Level 2 models. Level 3 is where decisions have to be made about the mechanisms to be used to effect the adaptations so it is the most obvious point where requirements start to merge with design. However, the Level 1 and 2 models may also be developed to the point where features, components or services of the implementation platform are identified. Clearly, this may be done in an iterative way using several cycles through the set of models.

To complement the goal modeling used for analysis of the DAS requirements, we have developed a domain specific language and associated tool support called *Genie* [1] that supports the transition to design. *Genie* is a visual tool that implements a domain-specific language for adaptive middleware frameworks. Adaptive middleware is a class of technology that has been developed with the express purpose of supporting DASs. By using the requirements that emerge from the Level 1-3 i^* models as input to *Genie*, a DAS that satisfies the requirements can be designed using a model-driven approach. Generative programming may also be possible depending on the adaptive middleware framework chosen. In the case of *Genie*, configurations of components from the GridKit [15, 18] component framework can be defined, thus providing traceability from goals all the way to code.

In the next section we use the GridStix case study to illustrate our approach.

3. Case study: The GridStix Flood Warning System

Flood warning systems play a key role in providing flood protection. Many existing flood warning systems use on-site wireless sensor networks (WSNs) to collect data. The data is typically transmitted off-site to be processed by computationally-intensive predictive models. Flood warning systems are often located in remote areas and have to use low bandwidth cellular network technologies for data transmission, hence limiting the volume and frequency of data transmission.

However, new embedded hardware and wireless networking technologies are enabling hydrologists to develop flood warning systems capable of performing local computation. Although constrained by size and power supply, a new generation of sensor network nodes have been developed with sufficient CPU power and memory to perform useful computations.

The availability of local computation power has a number of benefits. For example, for flow velocity measurement, it enables digital cameras to be used to capture the movement of tracer particles in the water. Digicams generate large volumes of data but are cheap and contrast with the alternative means of flow measurement using ultrasound sensors. Ultrasound sensors generate small volumes of data that can be easily transmitted off-site, but are costly. Large digital images cannot be efficiently transmitted off site but the availability of local computational power makes this unnecessary.

In addition, modern sensor nodes can communicate using high-bandwidth wireless technologies (e.g. IEEE 802.11b) over the short distances that separate the nodes. The resulting sensor networks are thus able to act as lightweight grids, thus enabling the performance of local *point prediction*, that is, analyzing the behavior of a river at a single point.

From the point of view of our work, the biggest benefit of the new technologies comes from being able to adapt dynamically (i.e., software is removed and new software is uploaded) in response to changing river conditions. CPU clock speeds can be varied, network topologies can change, and nodes can switch between different wireless communication technologies as the requirements for energy conservation, real-time computation, and fault-tolerance vary according to the state of the river.

GridStix [19] is an example of a flood warning system that embodies all of the above features. It has recently been deployed in prototype form on the flood plain of the River Ribble in North Yorkshire, England. The designers of GridStix needed to identify the requirements for a flood prediction system and use them to derive a solution design. In the remainder of this section we describe how our requirements model-driven approach was used to in GridStix.

3.1. GridStix Goal Identification

The first stage in the analysis process was to determine GridStix’s goals with a Stakeholder Goal Model. The i^* strategic dependency graph used to develop the Stakeholder Goal Model is shown in Figure 1. There are two actors represented by circles, a single goal represented by the round-cornered rectangle and three softgoals represented by the clouds. The ‘D’s on the arcs represent dependencies. The agents are the Environment Agency (the UK organization charged with managing the environment) and the GridStix system itself. The Environment Agency depends on GridStix to satisfy the goal Predict flooding which is the fundamental service that GridStix needs to offer. The three softgoals that were identified are: Prediction accuracy to avoid warning failures and false alarms; Energy efficiency because the deployed sensors have to depend on batteries or solar pan-

els for their power supply; and Fault tolerance because the sensor nodes are vulnerable to damage and may fail.

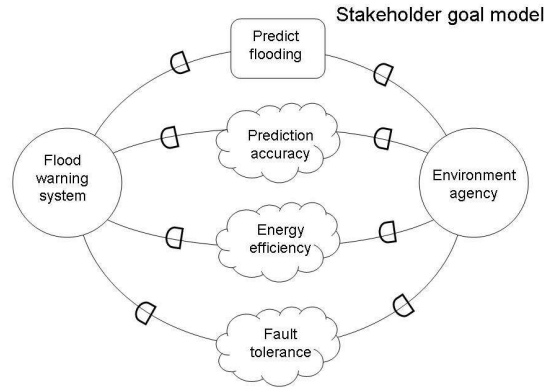


Figure 1. Strategic dependencies in the Grid-Stix domain

Next, the analyst worked with a hydrologist to identify states of the river environment, each of which could be treated as a discrete domain for a target system. The river states identified were: *quiescent* in which the river flow rate and water depth are within ranges that indicate that the river poses no threat to local or down-stream communities; *high flow* where the river flow rate has increased beyond a threshold value that could indicate that a significant and potentially damaging increase in water depth is about to occur; and *in flood*. Flooding results in inundation of the flood plain, damage to property and danger to communities and livestock. Node failure is probable when the river is in flood. Each river state corresponds to a target system of the DAS: S1: Normal; S2: Alert and S3: Emergency, respectively.

Having identified the river states and the corresponding target systems, next stage was to develop i^* models at each of the three levels of analysis, as shown in Figures 2 to 6. Here, the dotted lines indicate the scope of responsibility of the actor depicted in the circle that the line bisects. The hexagons represent tasks and square-cornered rectangles represent resources used or created. Tasks represent the means by which a goal or softgoal (the ends) may be satisfied. Tasks and goals may be decomposed into sub-tasks and sub-goals. Hence each model represents a specification of the goals and subgoals an actor has to satisfy and the tasks the actor may perform and the resources they may employ to achieve goal satisfaction.

3.2. Level 1 Models

The i^* strategic rationale models developed for each GridStix target system, S1, S2 and S3, are depicted in Figures 2, 3, and 4, respectively. Each model specifies how the stakeholder goals and softgoals are satisfied. Each Figure is of course the end result of reasoning about S1, S2 and S3 by developing a number of alternative configurations of goals and tasks in i^* .

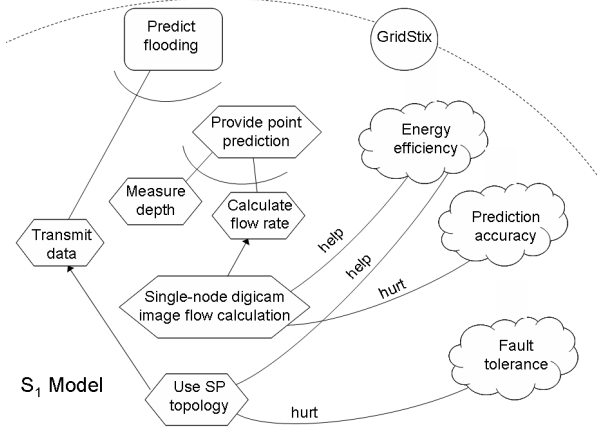


Figure 2. Level 1: S1: normal operation

For S1 Normal (Figure 2), the goal Predict flooding is satisfied by the tasks Provide point prediction, so that the data can be processed locally by lightweight point prediction models, and Transmit data, so the models can be used for alerting the emergency services. Provide point prediction does not only specify a key element of GridStix’s functionality. It also specifies the environmental variables to monitor that will be used at Level 2 to make adaptation decisions. Provide point prediction is decomposed into tasks Measure depth and Calculate flow rate which are the monitored environment variables. Calculate flow rate is decomposed into the task Single-node digicam image flow calculation. Transmit data is satisfied by the task Use SP topology, which specifies that a shortest path spanning tree algorithm should be used for routing data around the sensor network.

Construction of the Level 1 models revealed a conflict among the softgoals identified in the Stakeholder Goal Model. Energy efficiency is not easily reconciled with Prediction accuracy and Fault tolerance. For S1, Energy efficiency was considered to have a higher priority than Prediction accuracy and Fault tolerance. This was resolved by using single-node flow measurement which provides a less accurate prediction capability than processing an array of sensor data, but is more energy-efficient. When the river is quiescent, single-node flow measurement was judged to provide adequate forewarning of a change in river state.

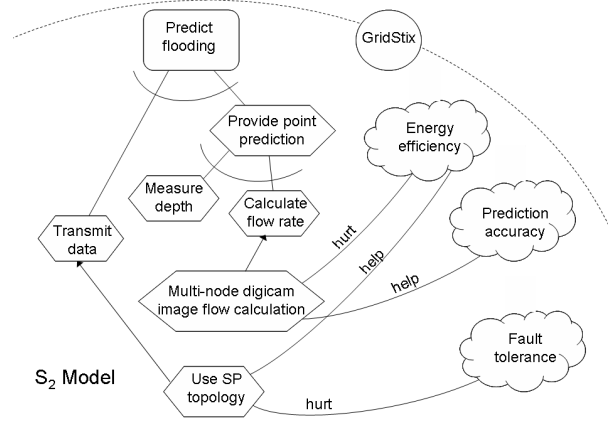


Figure 3. Level 1: S2: flow increase

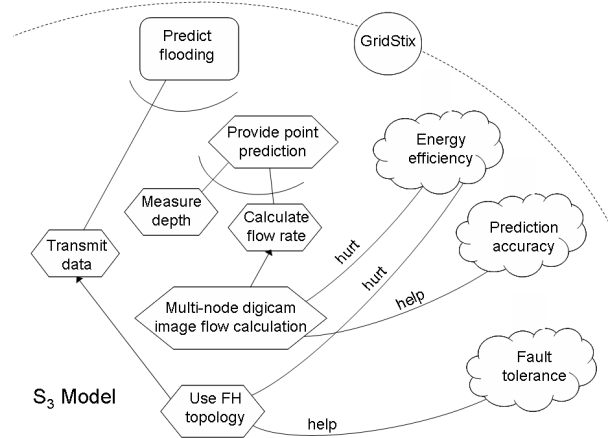


Figure 4. Level 1: S3: depth increase

Similarly, with the river quiescent, there is little risk of node failure so resilience was also traded off against low energy consumption. This was the rationale for specifying a relatively efficient shortest path network topology. These softgoal trade-offs are reflected in Figure 2 by the hurt and help relationships with the softgoals.

A different balance of trade-offs was used among the softgoals for S2 and S3. In S2 (Figure 3), an increase in flow rate often presages an increase in depth, so prediction accuracy is strengthened at the expense of energy efficiency by specifying Multi-node digicam image flow calculation. In S3 (Figure 4), the water depth has increased to the point where nodes are threatened by submersion or debris so a fewest-hop spanning tree (Use FH topology) is specified for the network topology. Fewest-hop networks are more resilient, though less energy-efficient, than shortest-path network topologies. The result of this choice was to strengthen

Fault tolerance at the expense of Energy efficiency.

3.3. Level 2 Models

Following completion of the strategic rationale models for each of the target systems S1, S2 and S3, the adaptation scenarios were developed. These address the decision-making concern and are i^* strategic rationale models specifying GridStix’s adaptation in response to river state changes. The River Ribble can change from quiescent to high flow so one of the Level 2 models must specify how GridStix adapts from S1 (the source system) to S2 (the target system). Each Level 2 model has to satisfy a single goal representing the desired adaptation. In the case of the S1 to S2 adaptation the goal is, unsurprisingly, Adapt from S1 to S2 as depicted in Figure 5. To satisfy the goal, the model must specify the values of the monitored variables and the adaptation requirements. The monitored variable for S1 to S2 adaptation is the river’s rate of flow and the critical value is represented as a threshold value. This monitoring condition is specified by the task (Detect when flow velocity...). The adaptation requirement is that processing of the digicam image processes must be distributed amongst the network nodes (Distribute digicam...). Note that these tasks are consistent with the monitored environment variables identified for S1 and the need to satisfy the multi-node digicam image flow calculation requirement in S2.

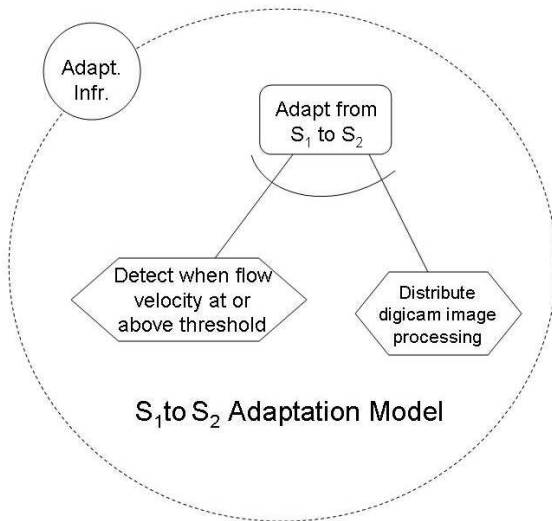


Figure 5. Level 2: S1 to S2 adaptation model

3.4. Level 3 Models

The final phase of the i^* modeling is addressing the adaptation concern by modeling the adaptation infrastructure (Figure 6). Only one adaptation infrastructure model is needed, but it is needed to ensure that the adaptation scenarios developed at Level 2 can be satisfied. We have identified 4 generic tasks that must be performed to satisfy the Enable Adaptation goal: Provide monitoring mechanism, Provide decision-making mechanism, Provide adaptation strategy and Provide adaptation mechanism. In Figure 6 these tasks are addressed by a set of resources. These are essentially place-holders for the technology to be selected. For GridStix, the adaptation infrastructure selected was GridKit [15, 18], a grid middleware technology that uses OpenCOM [4]. GridKit has an introspective capability which allows it to perform Architectural Reconfiguration using Component Substitution at run time. This adaptive behavior is defined by sets of Policy Rules and a Context Engine. GridKit rules that specify behavior in different contexts satisfy Level 1 requirements while rules that specify how the GridKit middleware adapts satisfy Level 2 requirements.

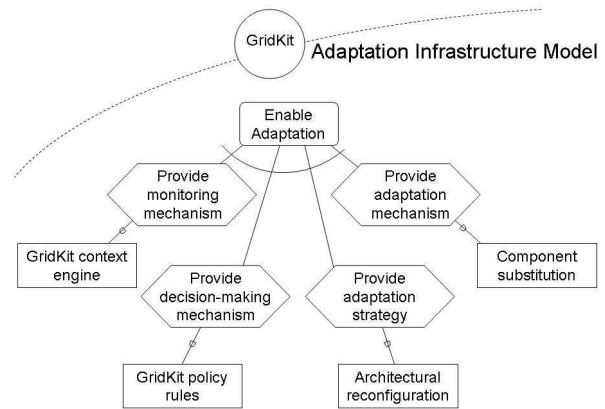


Figure 6. Level 3: adaptation infrastructure model

3.5. Where Requirements Meet Design

It is arguably not economical to build modern systems using custom code from scratch, and particularly so of DASs. Instead it is better to employ components or services that insulate the DAS application programmer from the complex system-level interaction with the operating platform hardware and software that is frequently necessary to meet the DAS requirements. In practice, DAS component

frameworks such as GridKit that are aimed at embedded systems still require a high degree of specialist system-level programming. One proposed solution is to define domain-specific languages [7] for the specification of run-time adaptation to increase the level of abstraction. Such languages are still at an early stage of development, but we have defined one for GridKit, and a visual tool called Genie [1] that supports it.

We do not have space here to present the DSL or all of Genie’s features. However, a key element of the design process that uses the DSL is the development of a state model that specifies how, and in response to which triggers, target systems adapt. A screenshot of the Genie model that specifies the transitions between GridStix target systems is shown as the diagram in the bottom right-hand corner of Figure 7.

Each node represents a target system specified in the Level 1 models. The target system nodes include configuration information derived from the Level 1 models. Hence, as specified in Figure 2, an efficient but (relatively) fault-intolerant spanning tree network topology is used in S1. Figure 2 also specifies Single-node digicam image flow calculation. Derived from this requirement, and the consequent low data transmission requirement, the low bandwidth but low energy-consuming BlueTooth (BT) wireless network technology has been specified as the networking technology for S1 in Figure 7.

The arcs refine the behaviour specified in the Level 2 models using the technologies specified in the Level 3 model. Adaptations such as those from S1 to S2 are modeled as state transitions. These correspond to the monitoring conditions specified in the Level 2 models. Recall that GridStix needs to detect when the flow rate reaches a threshold value. In Figure 7, this is modeled as the event High—Flow.

Derivation of the Genie target system adaptation model is assisted by the fact that we have refined the Level 1-3 models to the point where we have been able to make informed design decisions, and to select GridKit as the adaption infrastructure with which to implement GridStix. This has permitted the clean mapping from the Level 1-3 models onto the Genie target system adaptation model. As described above, requirements analysis of GridStix was application-driven using goal analysis, but as analysis proceeded, the i^* strategic rationale models were developed to the point where solution technologies could be identified. Hence, for example, the a shortest path network topology for S1 was selected in the knowledge that GridStix provides a component that implements a shortest path spanning tree. Our three-level, goal-driven approach is independent of GridStix but the selection of GridStix helped ease the transition to design. Crucially, since a GridKit-supporting DSL had been defined, the selection of GridKit enabled us to continue with a model-driven approach using Genie and

the clean mapping between specification and design models depicted in Figure 7.

Figure 7 also shows examples of GridKit policy rules that are invoked when the specified monitoring conditions are met. These policy rules are invoked at runtime and are used by GridKit to define the conditions under which a re-configuration of the architecture occurs. Hence, if, when GridStix is operating as normal, the point prediction model predicts an imminent flood, GridStix can adapt to the Emergency target system, by-passing the Alert target system. This adaptation is effected if the FloodPredicted monitoring condition is true by reconfiguring the overlay network to use *WiFi* instead of *BlueTooth*, and the spanning tree to a *FewestHop* topology.

Inevitably, we have only had space to show a subset of the models for GridStix, and the modeling features provided by Genie, and the analysis has necessarily been simplified. We have not shown the specification of tactics to enable sub-merged nodes to continue transmitting data, for example. Nor have we shown the component configuration models or any of the other models supported by Genie. However, we have described the principal goals, softgoals, and tasks that were used to drive the specification of the system and shown how they were used to derive lower-level requirements and inform the selection of an adaptation infrastructure. Figure 7 shows how combining our three-level analysis models with the Genie-supported DSL for adaptive middleware provided a visual, model-driven approach to GridStix’s development in which traceability was achieved, from high-level goals right through to the rules that defined the GridStix’s architecture.

4. Related Work

There is an emerging consensus among members of the requirements engineering research community who are interested in DASs, that goal-oriented approaches can be usefully deployed to model DAS requirements.

Goal-oriented specifications have been used to develop run-time monitoring devices. Specifically, Fickas *et al.* [10] have specified the requirements for an adaptive system in English and then used these requirements as guidelines for a run time requirements monitor. Additionally, Fickas *et al.* used the KAOS specifications to create Flea [5] event monitors. Feather *et al.* [8] continued in this spirit and specified how to model alternative behaviors of an adaptive system as alternate OR-refinements in a KAOS diagram [6].

Mylopoulos *et al.* [21, 29, 30] developed a process for inferring design information from annotated goal-oriented specifications, specified in a hybrid goal-oriented modeling technique using concepts and notations from KAOS and i^* [28].

Most recently, Fickas *et al.* [9, 11, 26] have proposed the

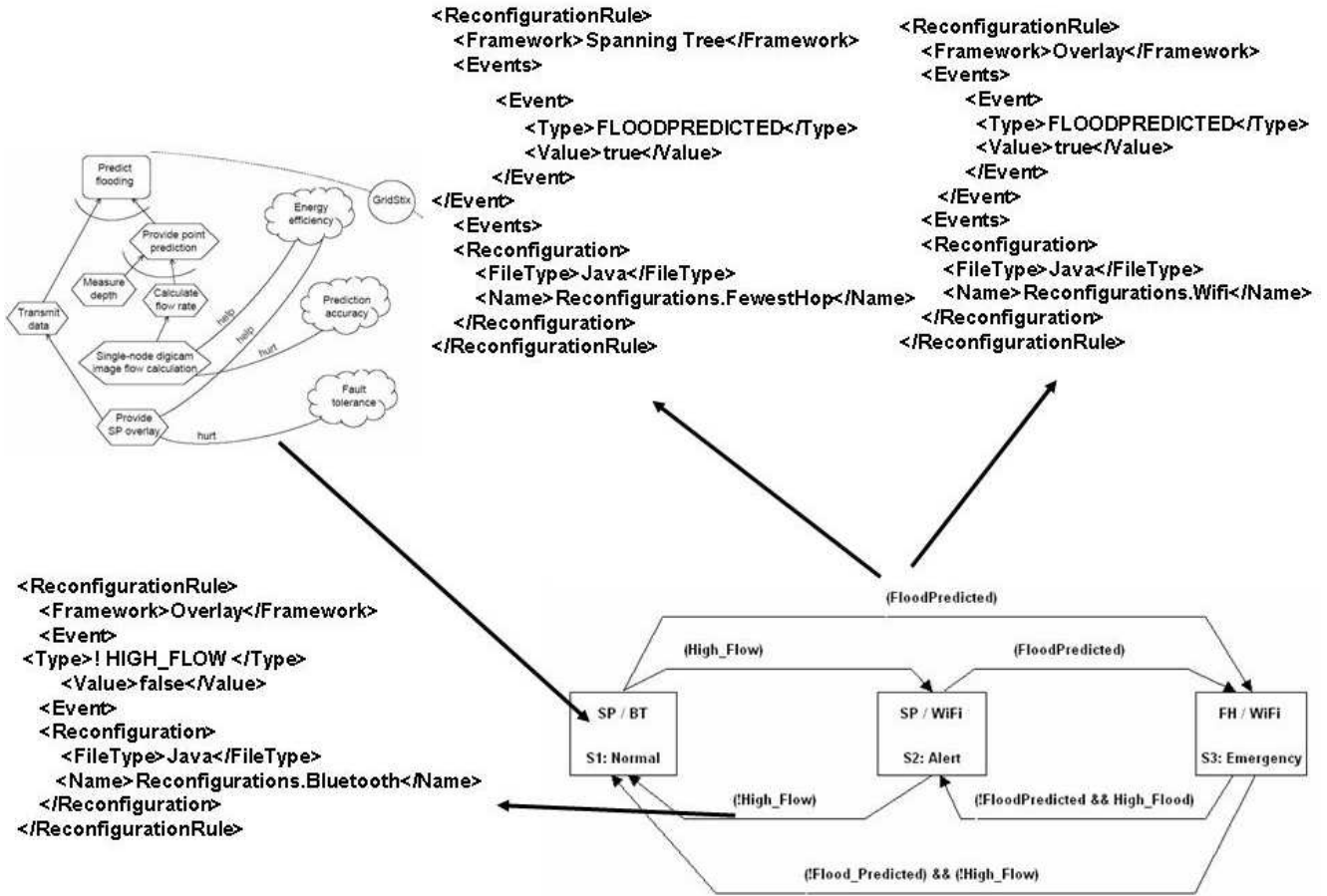


Figure 7. Genie model for transitioning between target systems showing traceability from analysis models to GridKit policy rules

idea of personal RE in which a system is designed to adapt to a particular user whose individual goals are specified in a goal attainment scale [9, 11].

These techniques all focus on specifying the conditions to monitor for adaptation, but do not [explicitly] model the other two RE concerns, adaptations and decision making. The advantage of our approach is that we use the four-level model of Berry *et al.* [3] to achieve a separation of concerns in which adaptation and decision making are represented explicitly.

There is a nascent research community [2, 12] that is exploring the potential for models to drive automatic system reconfiguration at run time. Our work, and the opportunities it illustrates for mapping from system goals to run time adaptation mechanisms serves as an early demonstration that run time model-driven engineering is feasible. Our vision is to make requirements the drivers of these run time models.

There is also a need for new techniques to help systems developers identify candidate target systems that handle the known, potentially adverse conditions and are sufficiently resilient and robust to handle slight variations of the adverse conditions. Recent work at Michigan State University is exploring how biologically-inspired techniques can be used to generate models for target systems more resilient than human-generated behavioral models [14].

5. Conclusions

This paper has presented an approach to modeling the requirements of dynamically adaptive systems (DASs). Our approach is intended to help cope with the complexity inherent in DASs by imposing a separation of concerns which clearly separates the requirements for a DAS to operate when its environment is stable, from the requirements for the DAS to adapt when its environment changes. Our ap-

proach provides a framework that allows the analyst to represent the key requirements of a DAS, including conditions to monitor, decision-making procedure, and possible adaptations.

We illustrated our approach with a case study based on GridStix, an adaptive flood warning system. Hitherto, designers of DASs such as GridStix have had very little support for analyzing their problem domains and understanding how the capabilities of their novel hardware and system software could support their adaptation requirements. Our approach provided some guidance for the design phase of development. GridStix's behavioural complexity is implemented using (re)configurations of components that, to a degree, insulate the application developers from system-level details of the hardware platform, the low-level features of which it needs to exploit.

There is a good mapping from the adaptation scenarios defined for Level 2 onto policy rules that define how GridStix reconfigures itself at run time. There was a similar mapping from the monitoring conditions defined at Level 1 to the context engine used by GridStix. Traceability from requirements to the policies defining GridKit's reconfigurations was therefore quite explicit. This was further enhanced by using the results of the i^* analysis as the input to a domain-specific language optimized for adaptive component frameworks and supported by the Genie tool.

Since, currently, the only economic way to engineer DASs is with existing infrastructures such as GridKit [17] or DynamicTAO [20], UIC [24], or RAPIDware [25], we expect similar opportunities for exploiting traceability for the benefit of system redesign and evolution. Our work now needs to be evaluated in a wider range of scenarios. In particular, for problems where there are a large number of possible states for the environment, the need to specify every possible adaptation between target systems may represent a bottleneck. More research is needed to evaluate the extent to which adaptation scenario explosion is a problem that requires mitigation. We need to discover the scope of our approach and how to identify DAS problems domains for which it is effective and those for which it isn't.

Finally, we need to define a process model to turn our approach into a systematic method. To achieve this we are building on the the work reported in [13].

References

- [1] N. Bencomo and G. Blair. Genie: a domain-specific modeling tool for the generation of adaptive and reflective middleware families. In *6th OOPSLA Workshop on Domain-Specific Modeling*, Portland, 2006.
- [2] N. Bencomo, G. Blair, and R. France. Models@runt.time. workshop in conjunction with models / uml 2006, October 2006 2006.
- [3] D. M. Berry, B. H. Cheng, and J. Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ)*, 2005.
- [4] G. Blair, G. Coulson, J. Ueyama, K. Lee, and A. Joolia. Opencom v2: A component model for building systems software. In *IASTED Software Engineering and Applications*, USA, 2004.
- [5] D. Cohen, M. S. Feather, K. Narayanaswamy, and S. S. Fickas. Automatic monitoring of software requirements. In *ICSE '97: Proceedings of the 19th International Conference on Software Engineering*, pages 602–603, Boston, Massachusetts, United States, 1997.
- [6] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. In *IWSSD: Selected Papers of the Sixth International Workshop on Software Specification and Design*, pages 3–50, 1993.
- [7] A. v. Deursen, P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. *ACM SIGPLAN Notices*, 35(6):26–36, 2000.
- [8] M. S. Feather, S. Fickas, A. V. Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *IWSSD '98: Proceedings of the 9th International Workshop on Software Specification and Design*, 1998.
- [9] S. Fickas. Clinical requirements engineering. In *ICSE '05: Proceedings of the 27th International Conference on Software Engineering*, pages 28–34, St. Louis, MO, USA, 2005.
- [10] S. Fickas and M. S. Feather. Requirements monitoring in dynamic environments. In *RE '95: Proceedings of the Second IEEE International Symposium on Requirements Engineering*, page 140, York, UK, 1995.
- [11] S. Fickas, W. Robinson, and M. Sohlberg. The role of deferred requirements in a longitudinal study of emailing. In *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering*, Paris, France, 2005.
- [12] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In L. Briand and A. Wolf, editors, *Future of Software Engineering*. IEEE-CS Press, 2007.
- [13] H. Goldsby and B. H. Cheng. Goal-oriented modeling of requirements engineering for dynamically adaptive systems. In *IEEE International Requirements Engineering Conference (RE06)*, Minneapolis, MN, 2006.
- [14] H. J. Goldsby, B. H. C. M. P. K. Knoester, D. B. Cheng, and C. A. Ofria. Digitally evolving models for dynamically adaptive systems. technical report msu-cse-07-4 computer science and engineering, michigan state university, east lansing, michigan, january 2007. Technical report, Computer Science and Engineering, Michigan State University, East Lansing, Michigan, January 2007.
- [15] P. Grace. *Overcoming Middleware Heterogeneity in Mobile Computing Applications*. PhD thesis, Ph.D. Thesis, Lancaster University, 2004.
- [16] P. Grace, G. Blair, and S. Samuel. A reflective framework for discovery and interaction in heterogeneous mobile environments. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(1):2–14, 2005.
- [17] P. Grace, G. Coulson, G. Blair, L. Mathy, D. Duce,

- C. Cooper, W. K. Yeung, and W. Cai. Gridkit: Pluggable overlay networks for grid computing. In *Symposium on Distributed Objects and Applications (DOA)*, Cyprus, 2004.
- [18] P. Grace, G. Coulson, G. Blair, and B. Porter. Deep middleware for the divergent grid. In *IFIP/ACM/USENIX Middleware*, Grenoble, France, 2005.
- [19] D. Hughes, P. Greenwood, G. Coulson, G. Blair, F. Pappenberger, P. Smith, and K. Beven. Gridstix:: Supporting flood prediction using embedded hardware and next generation grid middleware. In *4th International Workshop on Mobile Distributed Computing (MDC'06)*, Niagara Falls, USA, 2006.
- [20] F. Kon, M. Roman, P. Liu, J. Mao, T. Yamane, L. Magalhaes, and R. Campbell. Monitoring, security, and dynamic configuration with the dynamictao reflective orb. In *2nd ACM/IFIP International Conference on Middleware*, pages 121–143, New York, 2000.
- [21] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu. Towards requirements-driven autonomic systems design. In *DEAS '05: Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software*, pages 1–7, St. Louis, MO, USA, 2005.
- [22] R. R. Lutz. Targeting safety-related errors during software requirements analysis. In *SIGSOFT '93: Proceedings of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 99–106, Los Angeles, California, United States, 1993.
- [23] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004. And Companion Technical Report <ftp://ftp.cse.msu.edu/pub/crg/PAPERS/survey-tr.pdf>.
- [24] M. Roman, F. Kon, and R. H. Campbell. Reflective middleware: From the desk to your hand. *IEEE DS Online, Special Issue on Reflective Middleware*, 2(2), 2001.
- [25] S. Sadjadi, P. McKinley, and E. Kasten. Architecture and operation of an adaptable communication substrate. In *9th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, pages 46–55, Puerto Rico, 2003.
- [26] A. Sutcliffe, S. Fickas, and M. M. Sohlberg. Personal and contextual requirements engineering. In *RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering*, Paris, France, 2005.
- [27] V. M. Systems. Dynamic adaptive radiotherapy. <http://www.varian.com/orad/drt000.html>.
- [28] E. S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *RE '97: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, Washington, DC, USA, 1997.
- [29] Y. Yu, J. C. S. do Prado Leite, and J. Mylopoulos. From goals to aspects: Discovering aspects from requirements goal models. In *Proceedings of the 12th IEEE International Conference on Requirements Engineering (RE 2004)*, pages 38–47, Kyoto, Japan, 2004.
- [30] Y. Yu, J. Mylopoulos, A. Lapouchnian, S. Liaskos, and J. C. Leite. From stakeholder goals to high-variability software design. Technical report csrg-509, University of Toronto, 2005.
- [31] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 371–380, New York, NY, USA, 2006. ACM Press.