Revisiting the Relationship between Software Architecture and Requirements: the case of Dynamically Adaptive Systems

Nelly Bencomo, Paul Grace, Pete Sawyer
Computing department, InfoLab21, Lancaster University, LA1 4WA, United Kingdom
nelly@acm.org, gracep@comp.lancs.ac.uk, sawyer@comp.lancs.ac.uk

Abstract

This paper revisits the relationship between software architecture and requirements focusing on the case of self-adaptive systems. The authors present their view of the state-of-the-art, including their own work, on both areas and their contribution towards the development of self-adaptive systems. The authors support the claim that there is no fundamental distinction between architectural decisions and architecturally significant requirements and discuss how these claims are specifically appropriate for the case of self-adaptive systems. A discussion of the approach described and challenges for the case of adaptive systems are also presented.

Keywords*: architecture, requirements, dynamically adaptive systems.*

1. Introduction

For a long time the relationship between software architecture and requirements has been subject to deliberation. Several venues have hosted the exchange of ideas on the topic: e.g. the panel on the topic of Role of Software Architecture in Requirements Engineering at the first IEEE International Conference on Requirements Engineering (ICRE'94) and the workshop series "Software Requirements to Architecture" (STRAW 2001 and 2003). The relationship between Requirements and Architecture was also acknowledged by *Nuseibeh* & *Easterbrook* in [15]. These research efforts have produced different results with different visions of the relationship between requirements and architecture. For example, *Grünbacher et al.* [10] presented an approach to provide a systematic way of reconciling requirements and architectures. *Jackson et al.* [11] uses problem frames to allow architectural structures to be considered as part of the problem domain. In contrast to efforts that emphasize the gap that needs to be bridged between requirements and architecture, Nuseibeh [14] emphasizes "*the equal status we give to requirements and architectures*". He acknowledges a process that allows both requirements engineers and software architects to work concurrently and iteratively to specify the artifacts to be produced.

More recently, *de Boer* and *van Vliet* [7] argue that there is no fundamental distinction between architectural decisions and architecturally significant requirements. According to *de Boer* and *van Vliet* even though, software architectures and requirements engineering have different perspectives [s]"*oftware architecture is not merely the domain of the architect. Each architecturally significant requirements is already a decision that shapes the architecture"* [7]. Similarly, *van Lamsweerde* [17] claims that when using goal-oriented requirements engineering (GORE) high-level architectural judgements are made in that process. We claim in this paper that the arguments presented in [7] and [17] are particularly appropriate for the case of self-adaptive systems. Furthermore, and as in [7], we also agree the need to identify research areas in which tighter collaboration between the software architecture and requirements engineering research communities would lead to advantages and faster progress for both.

In this paper we present a systematic, incremental approach to deriving software architectures and their reconfigurations from system goals. Our approach is grounded on goal-based models with the intent of guiding software architects in their design task when developing self-adaptive systems. Specifically, our work on goal-based modelling supports the mapping between system goals and their impact in the architecture and its reconfiguration during runtime adaptations. The approach allows us to articulate different research challenges in terms of capturing, maintaining and adapting the requirements of a self-adaptive system and their impact on the software architecture after the system itself has been deployed and is operational.

2. State-of-the-art

The need of self-adaptation as a crucial enabling capability for many applications has stimulated researchers from both requirements engineering and software architecture research areas to propose different approaches. In the context of requirements engineering, *Berry et al.* [3] have proposed four levels of analysis that encompasses a framework of discourse for requirements of self-adaptive systems. A notable body of work has applied goal-based modeling notations, such as i* [9], to the discovery and

specification of requirements of self-adaptive systems. Goal-based models have proven to be effective for the specification of the adaptation selections that a self-adaptive system must perform and the specification of monitoring and change between adaptive behaviours [5, 17]. The ability to reason about partial goal satisfaction is a particular strength of goal-based modelling

Architecting self-adaptive systems has also produced a large body of work. A recent roadmap paper which discusses the state-of-the-art in software architecture for self-adaptive systems is given in [13]. This roadmap paper presents among many others the work of *Oreizy et al.* that introduced an architecture-based approach to self-adaptive software and evolution management; *Dashofy, van der Hoek* and *Taylor* propose the use of architecture-based evolution management for run-time adaptation using ArchStudio; *Garlan et al* have proposed the Rainbow framework that uses software architectures and a reusable infrastructure to support self-adaptation of software systems.

There have also been research efforts that combine both requirements and architectures to study the case of self-adaptive systems. Partially based on the research initiatives presented above, *Kramer* and *Magee* [13] describe their vision of self-management at the architectural level, where a self-managed software architecture automatically configures the interaction of its components to be compatible with an overall architectural specification and to achieve the specified goals of the system. *Floch et al.* [8] promote the use of architecture models to support the development of adaptive applications for mobile applications. In contrast to traditional event-action rules, *Floch et al.* propose the use of goal-based policies expressed as utility functions leaving to the system the decisions on the actions required to implement those policies. Partial results of our own research using requirements and architecture and that are explained in more detail in Section 3 are shown in [9, 16]. The approaches of *Kramer* and *Magee*, *Floch et al.*, and our own work carry the notion of the need to achieve the specification of goals in such a way that it is intelligible by the system itself and not only by humans. This last is an important point we visit again at the end of the paper when discussing the challenges.

It is common to find in the literature the explicit distinction between requirements and architecture. A decisive factor used in academia to distinguish both concepts encompasses "what" vs "how" and "problem" vs "solution". Industry is more pragmatic and that factor is usually related with "what" is determined "before" and "after" the contract is signed by the client. What is relevant to this paper, as our focus is on self-adaptive systems, is that the criteria described above encompasses the belief of "already fixed or agreed" vs "what remains to be done" [7]. " *A self-adaptive system is able to modify its behavior according to changes in its environment*" [5]. Requirements engineering for self-adaptive systems must deal with uncertainty [5, 18] because the information about future execution environments is incomplete, and therefore the requirements for the behavior of the system may change (at runtime) according to the changing environment. One of challenges that self-adaptation poses is that we cannot anticipate *requirements* for the whole set of possible conditions and their consequent impacts on the *architecture*.

3. Goal-based requirements specification for self-adaptive systems

Our requirements specification approach is goal-driven and characterizes the environment as a finite set of stable states subject to events that cause transitions between states. A self-adaptive system can be modeled as a collection of *target systems* [20], each of which correspond to, and operate within, a state of the environment [9]. The concerns modelled correspond to levels of analysis that represent particular concerns of a self-adaptive system: the behaviour of the set of target systems, the requirements for how the self-adaptive system adapts from one target system to the next, 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 and analyze the system requirements. Level 1 is analogous to the analysis needed for a conventional, non-adaptive system. A separate level 1 model is needed for each stable state of the environment for each target system. Level 1 models 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 prompt transitions between stable states, specify the target systems that represent the start and end points of adaptations and specify the form of the adaptation. Level 3 analysis is concerned with identification of the adaptive infrastructure in order to enable self-adaptation. Level 3 is not relevant for this paper.

We have successfully applied our approach to GridStix [12], an adaptive flood warning system and is documented in [9]. We describe only a subset of the the models for the case study. A fuller description is found in [9, 16]. The first task at level 1 was to identify the high-level, global goals of GridStix: the main goal Predict Flooding, and three goals, or *softgoals* that describe required qualities, Fault Tolerance, Energy Efficiency and Prediction Accuracy. Next, states of the river environment were identified, each of which could be treated as a discrete domain for a target system. In GridStix, these represented a three-fold classification of the river state: S1: Normal or quiescent, where depth and flow rate are both within bounds that indicate no risk of flood; S2: Alert where

the depth was within acceptable bounds but the flow rate had increased significantly, possibly presaging the onset of the next state; S3: Emergency.

The next stage in the analysis was aimed at discovering the application requirements for each target system. This necessitated development of an i* strategic rationale model (SR model) for each of the three target systems. SR models help the analyst reason about the requirements needed to address each environment variability. To illustrate this, consider the SR models for Normal state in Figure 1. The notation used is i*, target systems are depicted as agents represented by dashed circles, which depict the scope of the agents' responsibilities. The global goals and softgoals are Flow rate and Depth which are modeled as resources in the i* notation. Inside the agent boundaries, each target system is depicted as a set of tasks (the haxagones) that help to satisfy the Predict Flooding goal. The solid arrows arcs represent *means ends* relationships, while the arcs with bars at one end represent task decompositions. An open arrow arc represents a quantitative assessment of the extend to which a task contributes to the satisfaction of a soft goal. It can be annotated as hurt or help.

A key feature of i* is that it supports reasoning about softgoal trade-offs early in the analysis. The key aspect to note here is that the SR models allow the identification of tasks that specify the means by which goals are to be accomplished. These tasks corresponds to architectural relevant concerns (as also is the spanning tree explained below). According to the context, tasks may satisfy some softgoals better than others. As the river changes, the trade-offs between softgoals change and this impacts on the best combination of tasks (architectural decisions) to accomplish the Predict Flooding goal.

Construction of the Level 1 models revealed a conflict among the softgoals identified. Energy efficiency is not easily reconciled with Prediction accuracy and Fault tolerance. For S1, see Figure 1, 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. 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 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 S3:Emergency for example, 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

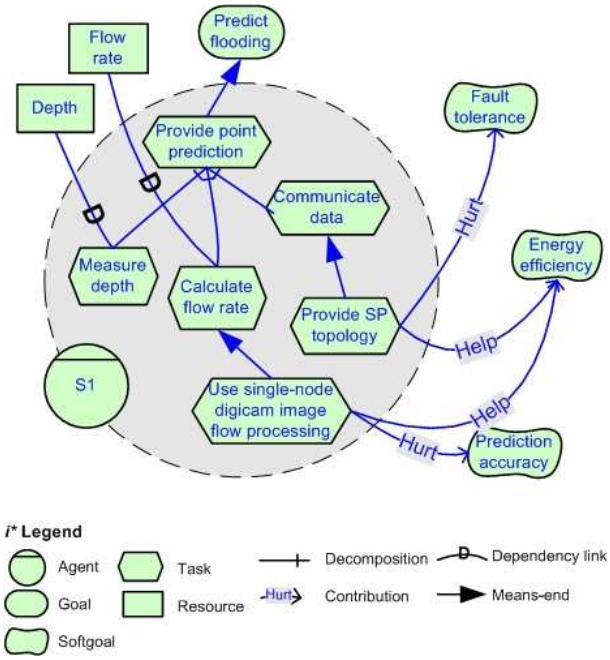

Figure 1. Behaviour model of environment variant Normal (from [9])

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. Similarly, multi-node digicam was chosen for this target system. See Figure 2).

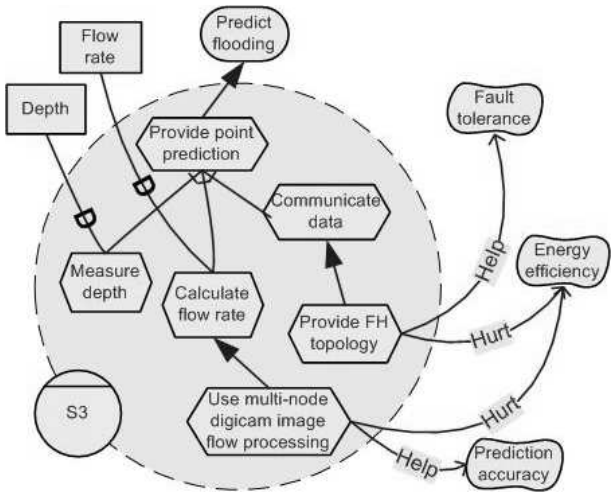

Figure 2. Behaviour model of environment variant Emergency (from [9])

Note how the trade-off between the conflicts of softgoals has required architectural decisions. Using SP or FH topology for the spanning tree or single-node or multi-node digicam for the image flow calculation respectively have proven to have different architectural impacts on the ongoing

system.

Similarly for S2: Alert a balance of trade-offs was used and the resultant strategy is shown in Figure 3).

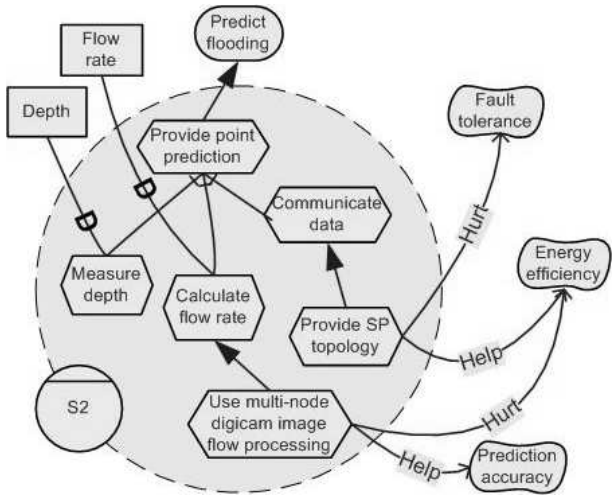

Figure 3. Behaviour model of environment variant Alert (from [9])

The Level 2 models identified and modeled adaptation scenarios that specify transitions between steady-state systems S1, S2, and S3. Figure 4 depicts the adaptation scenario for adapting from S1 to S2 as the river state transitions between the domains Normal and Alert.

In addition to specifying the source and target systems, each adaptation scenario must address three concerns that determine when and how to adapt: what data to monitor ; what changes in the monitored data trigger the adaption; and how the adaptation is effected. Each of these three concerns is conceptualized as the responsibility of a role of the adaptation infrastructure: Monitoring mechanism, Decision-making mechanism and Adaptation mechanism, respectively. The Decision-making mechanism, depending upon the source system S1, determines when GridStix needs to adapt from S1 to S2. This adaptation was specified to occur on a significant increase in river flow but before any significant depth increase. The Adaptation mechanism had to satisfy the goal Effect adaptation by performing the task Replace single-node flow processing with distributed flow processing and Replace SP tree with FH tree, which defined, at a high-level, the difference between the S1 and S2 behavior models.

Mapping from goals to architectural design

The Level 1 goal-based models described above have guided the design of the architectural-based models for each target systems. Similarly, the goal-based models associated with the transitions (models at Level 2) have guided the construction of models of the dynamic fluctuation of the environment and contexts, and their impact on the variation of the architecture of the applications during execution. Such architectural models have been constructed using the model-based tool Genie [1, 2]. Figure 5 shows a Genie model that specifies the transitions between the target systems (bottom right-hand corner).

Each transition (arc) describes when and how to dynamically switch from one target system to another (see details of the adaptation policies).The architectural concerns Routing Protocol and Image Processing will encompass the network topologies (SP and FH) and the singlenode processing (denoted as SC) and multiple-node digicam (denoted as MC). Each target system shows a pair of choices: Normal: (SP,SC), Alert: (SP,MC), and Emergency: (FH,MC). Furthermore, from the transition models, Genie allows the generation of different artifacts. e.g. the adaptation policies that will guide the adaptation of the system during execution, and the configurations of components associated with each target system. These artifacts can be dynamically inserted during runtime by the GridStix platform [12]. More information is found in [1, 2].

Combining our three-level analysis models with adaptive middleware platforms we have provided a model-driven approach for the development of the adaptive application GridStix. Traceability was achieved from highlevel goals right through to the adaptation policies that defined the GridStixs architecture [1]. From the goal-based models and the architecture decisions explained above the models for transitioning the system during execution and their policy adaptations were derived. Our approach is an example of how architecture decisions made during requirements specifications give early feedback and enable analysis. Our case study shows the benefits of close collaboration between requirement engineers and software architects for self-adaptive systems. The next section discusses such issues derived from our approach.

4. Discussion and Challenges

There is an increasing need for the requirements of a system to span requirements engineering activities during the development phases and runtime. So far, introduction of new requirements are not allowed during runtime using our approach. A self-adaptive system, who initially may employ a set of adaptation policies or strategies to meet the application's initial set of requirements provides a good example. As the application evolves the initial requirements may change. This may be due to a more accurate set of requirements being gleaned from the operation of the deployed system, or new capabilities being introduced into the system. Take for example the flood monitoring sensor network application explained above; initially the requirements in the development lifecycle produced a set of 6 adaptations between three software configurations [16]. The introduction of a new requirement (R1) may

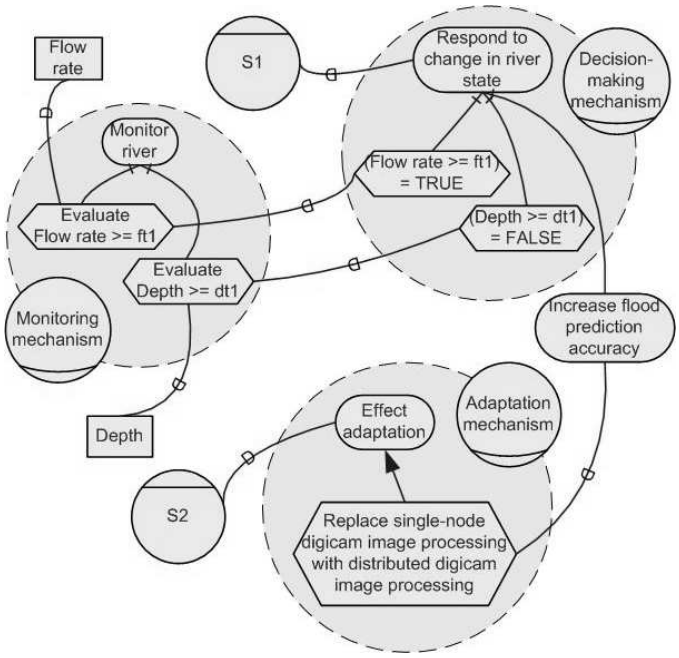

Figure 4. Level 2: S1 to S2 Adaptation Model

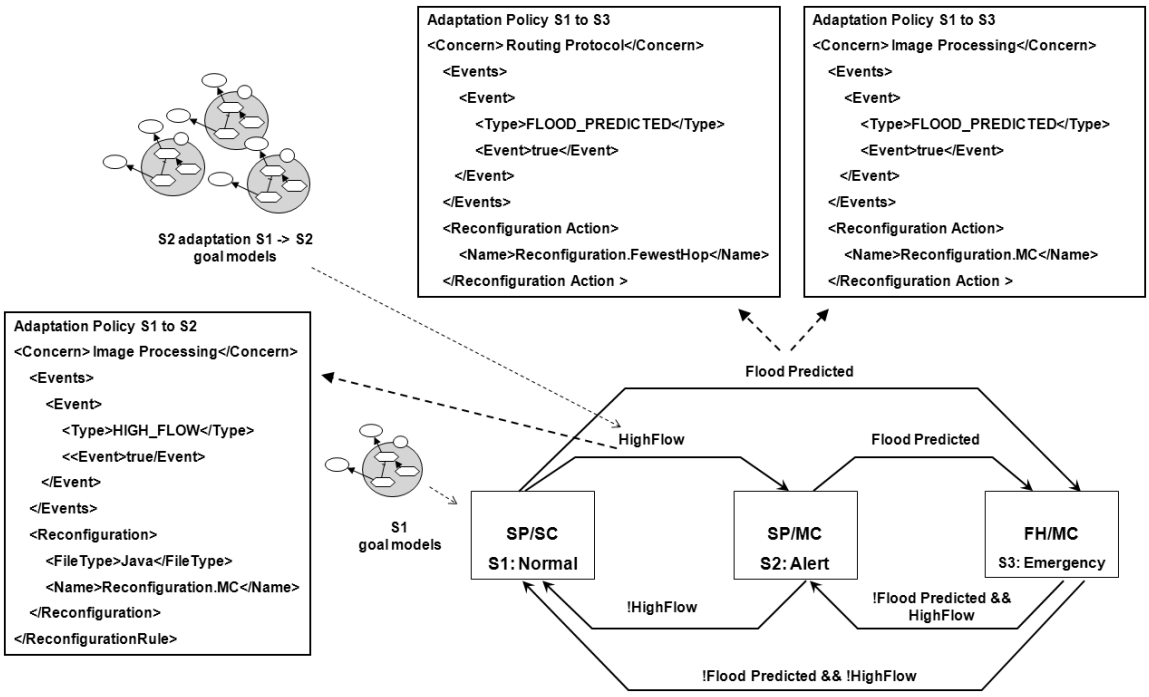

Figure 5. Genie models or transitioning between target systems showing traceability from the goal-based models to policy rules

introduce a new software configuration (a new target system) and transitions into the application, as well as affecting some of the other transitions. We can envisage that it would be appropriate to reconfigure the system to a target system of the form (fewest hops , single-node processing) i.e. (FH,SC), a target-system that was not foreseen and validated before. We are already working on a new requirements language called RELAX [18, 19] for self-adaptive systems that includes a vocabulary that enables requirements engineers to explicitly identify requirements that should never change as well as requirements that may change under certain conditions. Our final aim is to address uncertainty in requirements to support self-adaptive systems development in a way such that the uncertainty can be specified declaratively rather than by simply enumerating all alternative goals and respective architectures (in contrast to how it is currently done), some partial results are shown in [6]. Hence, this opens up various research challenges in terms of capturing, maintaining and adapting the requirements of a self-adaptive system after the system itself has been deployed and is operational. The new operators of a new RE specification language like RELAX should be traceable down to (dynamic) architectures at runtime. As in requirements management of uncertainty in architecture becomes a critical aspect of open evolution [4]. We posit that in order to allow requirements to be captured and dynamically changed at runtime, there are core challenges to be met.

Challenge 1. To create a model of a system's requirements that is maintained at runtime in order for it to be amenable to introspection and adaptation . Common to a reflective pattern, such an approach provides the following capabilities:
a) Introspecting the current requirements of the application allows: i) the performance of a system to be monitored to validate and verify that the operation of a self-adaptive system is meeting the requirements. ii) informing decisions for changing the requirements e.g. when requirements are no longer being satisfied, then they could be relaxed at runtime.
b) Adaptation of the application requirements. The requirements of an application can be changed at runtime through changes to the runtime model of requirements. These changes would be reflected in the physical, deployed architecture of the system.

Hence, further investigation is required into suitable models for representing requirements such that they remain available at runtime, and subsequently when changed they reflect on the system architecture.

Challenge 2. Identification and maintenance of a causal connection between the runtime model of requirements and the runtime model of the software architecture of the system. Such that changes in the software architecture can be monitored to ensure that the requirements are not broken; and also that changes to the requirements at runtime are reflected in the operation of running system through dynamic generation of changes to the software architecture artefacts.

Our vision is to enable a self-adaptive system to dynamically reason about (i.e. during execution) its own requirements and goals (a notion termed "requirements reflection" by Finkelstein [5]) to evaluate and directly affect the architecture and behaviour of the system. Software architectures should be flexible enough to be changed at runtime (evolve) [4] within certain constrains. What are the architectural concepts suitable for this new vision of requirements? What kind of constrains-aware operators can architecture offer to be leveraged by the system to meet new requirements at runtime? and to trade-off between the conflicts of soft-goals (NF requirements). Such questions seems to require tighter collaboration between requirements engineers and software architects because of the dynamic nature of self-adaptation.

Acknowledgment: This work was partially funded by the DiVA project (EU FP7 STREP).

References

[1] N. Bencomo and G. Blair. Using architecture models to support the generation and operation of component-based adaptive systems. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*. LNCS, 2009.

[2] N. Bencomo, P. Grace, C. Flores, D. Hughes, and G. Blair. Genie: Supporting the model driven development of reflective, component-based adaptive systems. In *ICSE 2008 - Formal Research Demonstrations Track*, 2008.

[3] D. Berry, B. Cheng, and P. J. Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *11th Inter Workshop on RE: Foundation for Software Quality (REFSQ'05)*, Portugal, 2005.

[4] S. Chaki, A. Diaz-Pace, D. Garlan, A. Gurfinkel, and I. Ozkaya. Towards engineered architecture evolution. In *Workshop MISE'09 at ICSE'09*, 2009.

[5] B. H. Cheng, H. Giese, P. Inverardi, d. L. e. a. R. J. A. Magee, Jeff, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. D. M. Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Muller, S. Park, S. Mary, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. *Software Engineering for Self-Adaptive Systems*, chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap. LNCS, 2009.

[6] B. H. Cheng, P. Sawyer, N. Bencomo, and J. Whittle. Goal-based modeling approach to develop requirements for adaptive systems with environmental uncertainty. In *ACM/IEEE 12th International Conference On Model Driven Engineering Languages And Systems , MODELS 2009*, 2009.

[7] R. C. de Boer and H. van Vliet. On the similarity between requirements and architecture. *Journal of Systems and Software*, 82(3):544 – 550, 2009.

[8] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. Using architecture models for runtime adaptability. *Software IEEE*, 23(2):62–70, 2006.

[9] H. J. Goldsby, P. Sawyer, N. Bencomo, D. Hughes, and B. H. Cheng. Goal-based modeling of dynamically adaptive system requirements. In *15th IEEE Conference on the Engineering of Computer Based Systems (ECBS)*, 2008.

[10] P. Grünbacher, A. Egyed, and N. Medvidovic. Reconciling software requirements and architectures with intermediate models. *Software and System Modeling*, 3(3):235–25, 2004.

[11] J. Hall, M. Jackson, R. Laney, B. Nuseibeh, and L. Rapanotti. Relating software requirements and architectures using problem frames. pages 137–144, 2002.

[12] D. Hughes, N. Bencomo, G. S. Blair, G. Coulson, P. Grace, and B. Porter. Exploiting extreme heterogeneity in a flood warning scenario using the gridkit middleware. In *Middleware (Companion)*, pages 54–57, 2008.

[13] J. Kramer and J. Magee. Self-managed systems: an architectural challenge. In *FOSE '07: 2007 Future of Software Engineering*, pages 259–268. IEEE Computer Society, 2007.

[14] B. Nuseibeh. Weaving together requirements and architectures. *Computer*, 34(3):115–117, 2001.

[15] B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 35–46, New York, NY, USA, 2000. ACM.

[16] P. Sawyer, N. Bencomo, D. Hughes, P. Grace, H. J. Goldsby, and B. H. C. Cheng. Visualizing the analysis of dynamically adaptive systems using i* and dsls. In *REV'07: 2nd Interl Workshop on Requirements Engineering Visualization*, India.

[17] A. van Lamsweerde. From system goals to software architecture. In *3rd Interl School on Formal Methods for the Design of Computer, Communication and Soft Systems*, 2003.

[18] J. Whittle, P. Sawyer, N. Bencomo, and B. Cheng. A language for self-adaptive system requirement. In *SOCCER Workshop*, 2008.

[19] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems". In *17th IEEE International Requirements Engineering Conference RE 2009*, 2009.

[20] J. Zhang and B. H. Cheng. Model-based development of dynamically adaptive software. In *International Conference on Software Engineering (ICSE'06)*, China, 2006.