From Awareness Requirements to Adaptive Systems: a Control-Theoretic Approach

Vítor E. Silva Souza, John Mylopoulos
Department of Information Engineering and Computer Science
University of Trento, Italy
{*vitorsouza, jm*}*@disi.unitn.it*

***Abstract*—Several proposals for the design of adaptive systems rely on some kind of feedback loop that monitors the system output and adapts in case of failure. Roadmap papers in the area advocate the need to make such feedback loops first class entities in adaptive systems design. We go further by adopting a Requirements Engineering perspective that is not only based on feedback loops but also applies other concepts from Control Theory to the design of adaptive systems. Our plans include a framework that reasons over requirements at runtime to provide adaptivity to a system proper. In this position paper, we argue for a control-theoretic view for adaptive systems and outline our long-term research agenda, briefly presenting work that we have already accomplished and discussing our plans for the future.**

***Keywords*-requirements; awareness; adaptive systems; feedback loop; control theory**

I. Introduction

With the increasing complexity of software systems, software engineering researchers and practitioners are now looking into self-adaptivity as a way to lower costs for the management of complex systems. Several proposals for the design of adaptive systems include some kind of feedback loop (e.g., the MAPE feedback loop [1]) that monitors the system's performance and triggers adaptation actions in case of failure.

In the book published after the Dagstuhl Seminar 08031 on "Software Engineering for Self-Adaptive Systems" [2], Brun et al. [3] notice that "while [some] research projects realized feedback systems, the actual feedback loops were hidden or abstracted. [...] With the proliferation of adaptive software systems[1] it is imperative to develop theories, methods and tools around feedback loops." Furthermore, Andersson et al. [4] consider that "a major challenge is to accommodate a systematic engineering approach that integrates both control-loop approaches with decentralized agents inspired approaches."

In our research, we take a Requirements Engineering (RE) perspective, starting with the question: what are the requirements that lead to feedback loop functionality? This question has led us to proposing a new class of requirements, called *Awareness Requirements*, which talk about the runtime status of other requirements [5].

[1]Throughout this position paper, we use the term "adaptive software system" to refer to systems that have mechanisms for monitoring, diagnosis and self-adaptation. Others call such systems "self-adaptive".

Furthermore, given that feedback loops are a central element of control systems [3], we began to explore techniques from Control Theory that could be useful in the design of systems that use feedback loops as a mechanism for adaptation. We again applied our requirements engineering perspective and proposed that *System Identification* — the process of determining the rules that govern a system's dynamic behavior — should be conducted during the modeling of an adaptive system's requirements, using qualitative information to deal with uncertainty [6].

The final goal we have envisioned for this research is a framework that uses the information gathered during *Awareness Requirements* elicitation and *System Identification* at runtime, adapting the target system in case its requirements are not satisfied. Moreover, we propose to develop a systematic process that starts with *Awareness Requirements* and goes all the way to a running adaptive system based on this framework, preferably with tool support.

In this paper we present our position on a requirements-based control-theoretic approach for the modeling, design and implementation of adaptive systems. First, section II explains our view of adaptive systems as control systems; then, section III outlines our long-term research proposal, summarizing what we have accomplished thus far and discussing future directions; finally, section IV closes the paper.

II. Control-theoretic View of Adaptive Systems

In our research, we assume the architecture for the design of an adaptive system uses one or more feedback loops to implement adaptivity. In other words, we see adaptive systems as feedback control systems.

A. Control Systems

Figure 1 shows a simplified view of a control system, adapted from [7]. In this kind of system, the *reference input* is "the desired value of the measured outputs"[2], while the *measured output* is "a measurable characteristic of the target system". For instance, consider a (simplified view of a) car's cruise control mechanism, which is a classic example of a control system. Its purpose is to maintain the car at some constant speed S_I. In this example, S_I is the *reference input*,

[2]this and the following quotes were taken from [7], §1.1

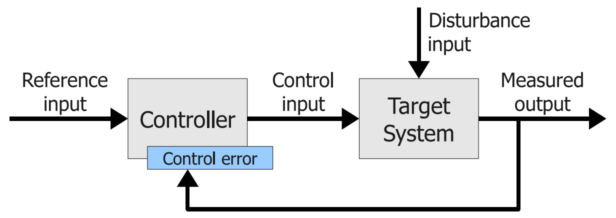

Figure 1. Simplified block diagram of a control system based on [7].

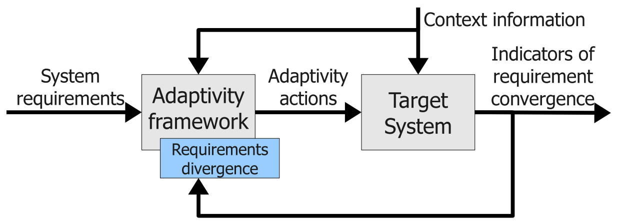

Figure 2. View of an adaptive system as a control system.

whereas the actual speed of the car S_O, which can be read from the car's speedometer, is the *measured output*.

Given this information, the *controller* "computes values of the control input based on current and past values of control error." The *control error* is "the difference between the reference input and the measured output," while the *control input* is "a parameter that affects the behavior of the target system and can be adjusted dynamically." Back to the example, the *control error* E can be calculated as $E = S_I - S_O$, leading to a straightforward definition for the *control input*: if $E > 0$, the *controller* (the cruise control system) should inject more fuel in the engine to speed up the vehicle (the *target system*). Analogously, if $E < 0$, less fuel should be injected. The idea is to keep S_O as close as possible to S_I at all times.

Finally, the *disturbance input* "are factors that affect the measured output but for which there is no governing control input." In other words, these are taken from the context in which the system executes. Neither the system nor the controller have any control over these values. For the cruise control system, the inclination of the road and the direction and strength of the wind are examples of *disturbance inputs*, as they can have an influence on the measured speed S_O.

B. Adaptive Systems as Control Systems

Our view of an adaptive system as a control system is represented in figure 2 and described in the following paragraphs. For *system requirements*, we adopt a Goal-Oriented approach which uses as modeling primitives concepts such as goals, softgoals, quality constraints and domain assumptions [8]. It is important to note, however, that our proposal starts at the *late requirements* phase (see, e.g., [9]), focusing on the requirements of the software system instead of the goals of the different agents that interact with it.

As a (classic) example to illustrate the definitions that follow, consider a meeting scheduler system[3] with goals such as *Collect timetables* from participants, *Find available rooms* and *Choose schedule*; domain assumptions like Participants use the system calendar for their work schedule; softgoals such as *Good participation* in meetings, which are more precisely defined by quality constraints like *At least 90% of participants attend* the scheduled meeting. A complete goal model for this example can be found in [6].

[3]We recognize the meeting scheduler is a very simple system and we have selected it for illustration purposes only. We are currently working on a larger case study to demonstrate the validity of our approach.

Reference input: in an adaptive system, the *reference input* consists of the *system requirements*. This includes not only "vanilla" requirements such as the system's goals and assumptions, but also adaptivity requirements. In section III we argue that *Awareness Requirements* and the information collected during *System Identification* are part of the system's adaptivity requirements.

Measured output: if requirements are the *reference input*, the *measured output* should then consist of *indicators of requirements convergence*. In other words, we would like to measure, at runtime, if functional requirements are being met (e.g., are users able to successfully *Find available rooms* for their meetings? What is the success rate for this goal?) and what are the degrees of satisfaction of non-functional requirements (e.g., what is the participation percentage for each scheduled meeting? Is it above 90%? What is the success rate for this quality constraint?). Such indicators are usually of Boolean nature (i.e., satisfied: true/false) and measures in other domains (e.g., the response time of a task or the success rate of a goal, both numeric) can be mapped to Boolean by a function that maps each value of the domain to satisfied/unsatisfied (in the case of numeric values, a threshold usually provides such mapping).

Following standard Goal-Oriented Requirements Engineering (GORE) approaches, we associate to every leaf-level goal one or more tasks through which the goal can be satisfied, and to leaf-level softgoals a quality constraint ("metric") for measuring the degree of satisfaction of the softgoal. A specification for a given goal model consists of a set of tasks, quality constraints and domain assumptions such that assuming that domain assumptions hold true, if tasks are executed and quality constraints hold, then all root-level goals are fulfilled. Of course, this is an optimistic view of the world. The tasks that are part of a specification may actually not be carried out during any one execution, or may not have the expected effects because of a fault. Also, quality constraints may be satisfied approximately, rather than fully. And domain assumptions may not hold in particular circumstances. For example, we assume all *Participants use the system calendar* in the meeting scheduler system, but is that really the case at runtime? This is useful information to monitor for when trying to satisfy the goal *Collect timetables*.

In section III, we defend our position that the evaluation of *Awareness Requirements* at runtime can provide such *indicators of requirements convergence*.

Control error: given the above *reference input* and *measured output*, the *control error* consists of a set of requirements (be they goals, quality constraints or even domain assumptions) that were not satisfied either individually (i.e., during a single execution of the system) or in an aggregate way (average success rate). Negative answers to questions presented previously (Are user able to successfully use the system? Is participation above 90%? Are 95% of all meeting participants using the timetable database?) are examples of *requirements divergence*.

Control input: given the information on the *control error*, the *control input* consists, of course, of the adaptivity actions, which might include reconciliation of system behavior and compensation to avoid inconsistent system states. In section III, we advocate that the information obtained from *System Identification* [6] is very useful for determining the *control input*.

Disturbance input: the factors that can be measured but that neither the *target system* nor the *adaptivity framework* have any control over are called *context information*. However, unlike the *disturbance input* in control systems (figure 1), *context information* in adaptive systems (figure 2) are provided as input not only to the *target system*, but also to the *adaptivity framework*. The reason for this is that the controller itself can be context-sensitive, selecting appropriate *adaptivity actions* depending on the context. At this time we have not addressed this issue with the necessary depth and regard the impact of contexts in our approach as future work.

It is with this control-theoretic view that we propose a systematic approach for the modeling, design and development of adaptive systems, based on feedback loops. We discuss this next.

III. The Approach

The long-term objectives of our research consist of developing:

- A systematic process that covers requirements engineering, design and implementation of adaptive systems based on a control-theoretic perspective;
- A framework that implements generic adaptivity actions on any instrumentable software system, given suitable requirements models;
- A CASE tool that guides requirements engineers through the steps of the aforementioned process that are related to adaptivity requirements. In other words, the tool would help create the proper models that are required as input for the adaptivity framework.

Figure 3 illustrates the proposed software development process for adaptive systems. In the following we discuss different aspects of this process.

A. *"Vanilla" and "Adaptive" Software System Development*

Figure 3 divides the software development process in two tracks: *"Vanilla" Requirements Engineering, Design and Coding* is at the top, while *Requirements Engineering for Adaptive Systems* can be found at the bottom. This separation of concerns is merely conceptual, not processual, i.e., the adaptive part of RE is, of course, highly dependent on the (partial or final) results of the "vanilla" RE activity and they are not parallel, independent subprocesses, as the diagram might suggest.

The "vanilla" software development process is concerned with modeling the requirements, designing the architecture and coding the target system in the usual way, i.e., independently of aspects related to adaptivity capabilities that the target system is supposed to have. For these activities, we do not prescribe any specific process or methodology.

However, we do impose a few constraints on this process, all related to our proposals on *RE for Adaptive Systems*, discussed in the next subsections. These constraints are:

- Our framework is goal-oriented and expects as output of the "vanilla" RE phase a goal model of the system's requirements. In particular, we have been basing our implementations on the RE ontology of Jureta et al. [8];
- In order for the feedback loop implemented in the *adaptivity framework* to perform run-time reasoning over the requirements model, the *target system* should log (in a medium accessible by the framework) information about the execution of system requirements. If traceability links were formalized during the development process, instrumentation techniques could be used to provide such logging;
- Finally, to adapt the *target system*, the *adaptivity framework* should be able to manipulate *parameters* of the system, which should also provide some callback functions such as `abort(R)`, `initiate(R)`, `retry(R)`, etc., where `R` is a requirement.

It is important to note that the process described in this section does not support legacy systems. Adding adaptivity capabilities to existing systems for which models and/or source code are not available is considered future work in the context of this research.

Given that the "vanilla" software development process has provided the required artifacts described above, we propose a systematic process for the inclusion of adaptivity features in the system-to-be. This process is composed of three main activities: *Awareness Requirements Elicitation*, *System Identification* and *Adaptation Strategy Selection*.

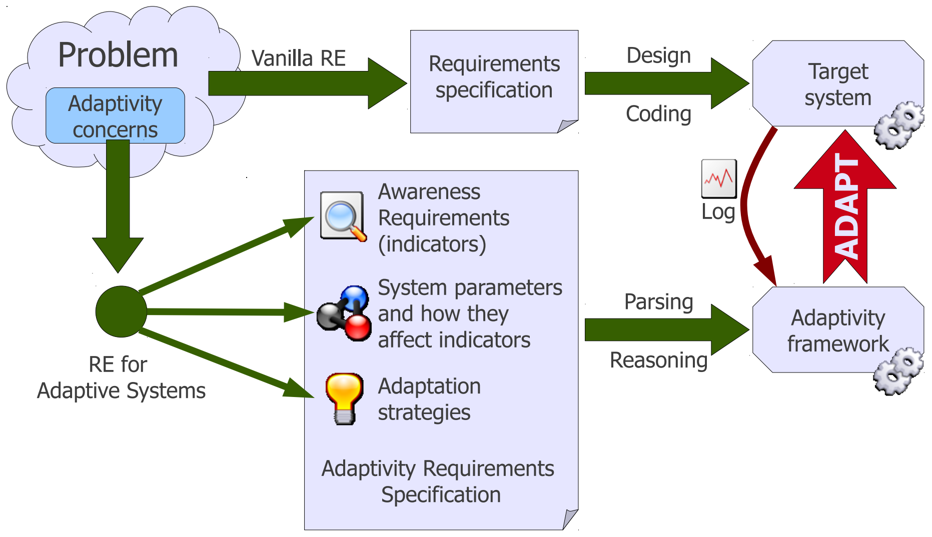

Figure 3. Overview of a software development process for an adaptive system.

B. Awareness Requirements Elicitation

As we have seen in section II, the feedback loop begins with the measurement of system output or, in the case of adaptive systems, the *indicators of requirements convergence*. In essence, we would like to know if requirements have been satisfied, denied, canceled, etc. For this reason we have proposed a new type of requirement called *Awareness Requirement* [5].

Awareness Requirements (*AwReqs*) are requirements that talk about the success, failure or any other possible change of state of other requirements — goals, quality constraints, domain assumptions and *AwReqs* themselves. Considering the meeting scheduler, examples of *AwReqs* could be:

- Domain assumption *Participants use the system calendar* should always be true (in terms of requirements states, it should never fail);
- Quality constraint *At least 90% of participants attend* should have 75% success rate;
- Considering one week periods, the success rate of goal *Collect timetables* should not decrease three times consecutively.

AwReqs can, thus, refer to requirements at the instance level (a single execution), at an aggregate level (success rates) and even monitor trends in success rates of requirements. In [5], *AwReqs* are characterized with more detail and the tasks of elicitation and formalization of this new type of requirement are discussed. As validation, we show that *AwReqs* can be monitored at runtime.

Awareness Requirements provide the *indicators of requirements convergence* needed by the feedback loop to calculate the *control error* (*requirements divergence*). The next step, then, is to find out how we can make the *measured output* as close as possible to the *reference input*. In other words, what could be done to help the system meet its requirements after *AwReq* failures?

C. System Identification

To answer the previous question, we go back to control systems. Take the car's cruise control example once again: knowing the type of fuel and the engine characteristics (torque, horsepower, etc.) you can calculate with high precision how fuel injection affects the car's speed, given the environment conditions (weight of car, inclination of road, wind speed, etc.). In Control Theory, the process of determining the equations that govern this kind of system behavior is called *System Identification*.

However, for complex socio-technical systems[4] deployed in environments with high uncertainty, such white box models (in which all the variables are known and can be calculated) are overly complex or even impossible to obtain. For this reason, we propose the identification of qualitative relations between *indicators of requirement convergence* and *target system parameters* that can be tuned at runtime [6].

For example, suppose there is an *Awareness Requirement* on the meeting scheduler that says goal *Find available rooms* should never fail and that information on the *target system*'s log indicates this *AwReq* has failed. In this case, the *adaptivity framework* should know which *parameter* can be modified in order to improve the chances of requirements convergence with respect to this specific indicator. If the meeting scheduler has a *parameter* `RfM` which indicates the number of *Rooms for Meetings* available to the system,

[4]Systems which "include in their architecture and operation organizational and human actors along with software and hardware ones" [10]

increasing this *parameter* could be considered an adaptivity action here. Likewise, if the goal *Find available rooms* is OR-decomposed into subgoals *Find local rooms*, *Call partner institution* and *Call hotels and convention centers*, selecting *partner* instead of *local* or *hotel* instead of *partner* would also help. These same changes might also have impact on other indicators (e.g., the cost of the scheduling) and all the available information should be taken into consideration by the *adaptivity framework* when deciding the *adaptivity actions* that should be executed.

In [6], we propose a language for modeling qualitative information on the relation between changes in *parameters* and the *measured output* of the system, and a systematic process for conducting *System Identification* for adaptive systems. We believe this information is fundamental for determining the *control input* in a feedback loop-based adaptive system and is currently missing from requirements models.

D. Adaptation Strategy Selection

Given indications that the system's requirements are being satisfied or not (evaluation of the *Awareness Requirements*) and information on how to improve these indicators in case of failure (result of *System Identification*), the missing piece of our *adaptivity requirements specification* is the exact actions the *target system* should execute in order to adapt itself. This is called the *adaptation strategy* and it is our current work in the context of this research.

The *adaptation strategy* can be divided in two parts: compensation and reconciliation. **Compensation** means bringing the system back to a consistent state, i.e., in case the tasks executed when the failure was detected left any inconsistencies behind, the feedback loop compensates for their actions. Here, we intend to follow the model of database long lived transactions [11] as done by Wang et al. [12]: the requirements engineer attaches compensation actions to specific elements of the goal model, representing what has to be done in case that specific element is part of a configuration that has failed.

Reconciliation refers to reconciling the system's run-time behavior and its requirements [13]. Given the information modeled during *System Identification*, two reconciliation mechanisms can be derived and are considered the default *adaptation strategies* by the *adaptivity framework*:

- **Parameter tuning**: if there are *parameters* that can be modified to improve *requirements conformance* (given the specific requirement that failed), analyze the information about these *parameters* (i.e., how changing them affects other *measured outputs*) and choose the best course of action, possibly following some generic system policies (e.g., "aggressive", "conservative", etc.). We have not yet devised a precise algorithm for this end;
- **Abort**: if there are no *parameters* that affect the given *indicator* or if all the *parameters* that affect it cannot be further tuned (they are at their minimum/maximum values), tell the *target system* to gracefully fail or degrade performance. The *abort* strategy is the last resort and it could also be used in case the *parameter tuning* strategy runs into conflicting courses of action and cannot solve the conflict.

Other reconciliation strategies can be obtained from related work in the area of adaptive systems. For instance, proposals such as [14] and [15] divide requirements into *crisp* and *fuzzy/relaxed*. Inspired by such ideas, we could derive a **good enough** strategy that consists on relaxing some satisfaction criteria for requirements. For example, quality constraint *At least 90% of participants attend* could be relaxed and deemed satisfied if the average participation is between 85% and 90%.

At this point, our research could also possibly benefit from ideas from other fields of computer science that deal with adaptive systems, such as "fault-tolerant computing, distributed systems, biologically inspired computing, distributed artificial intelligence, integrated management, robotics, knowledge-based systems, machine learning," etc. [16]. Ideas from these and other fields could inspire different adaptation strategies and enhancements on our current *adaptivity requirements specification*.

E. Adaptivity Framework and CASE Tool

The process that we have presented and advocate for the development of adaptive systems should be supported by two software artifacts: the *adaptivity framework* and a CASE tools for adaptive systems modeling, analysis and design. The CASE tools would assist the requirements engineer in producing the *adaptivity requirements specification*, whereas the framework would be responsible for reading such specification, analyzing the *target system*'s log and executing *adaptation actions* following the strategies that have been selected.

Parts of this tool suite have been developed for the execution of experiments to validate the ideas presented on our previous works [5], [6]. Nonetheless, we intend to develop a complete tools suite that would allow a developer to start with *Awareness Requirements* and go all the way to a running adaptive system.

IV. Conclusions

It has been recognized by the research community (e.g., SEAMS [17]) that ideas from Control Systems should be adopted for the development of adaptive systems. In this position paper, we have argued for a Requirements Engineering perspective for the design of feedback loop-based adaptive systems and provided an overview of our development approach. In this approach, "vanilla" goal-oriented requirements specifications are augmented with *Awareness*

Requirements and information produced during *System Identification*. Then an *adaptivity framework* is responsible for analyzing this information and follow *adaptation strategies* to adapt the *target system* at runtime.

This is on-going research and we see already opportunities for future work on several questions. For instance:

- Control systems can use negative and positive feedback [3]. What is the role of positive feedback in adaptive systems?
- Our proposal adds reactive adaptation capabilities to systems, but an even more useful feature would be to prevent failures altogether instead of reconciling after they occur. Predictive or probabilistic reasoning could be useful here;
- As mentioned in section II-B, our research focuses on system models, whereas other proposals (e.g., [18]) work with autonomous agents with their own requirements. In this context, a possible *adaptation strategy* would be a change in the delegation of responsibility over a requirement that has failed to be satisfied;
- What is the role of contexts in our models? We have mentioned *context information* in figure 2, but our approach does not properly address this input. Adaptive systems could have different *Awareness Requirements* or *indicator–parameter* relation depending on context. Existing work on contextual requirements (e.g., [19], [20]) would be the baseline here;
- How could our approach help on requirements evolution? After the system has been running for some time, could we benefit from information on what adaptations were needed (and which ones couldn't be performed and resulted in the *abort* strategy being selected) to help developers evolve the system's requirements?
- As mentioned in section III-A, how could we adapt this approach to deal with legacy systems, i.e., systems for which models and/or source code are not available?

We hope this paper will help foster discussion on the subject, which (in the spirit of feedback loops) will ultimately lead to the improvement of the ideas contained herein.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, Eds., *Software Engineering for Self-Adaptive Systems*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 5525.

[3] Y. Brun, G. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, *Engineering Self-Adaptive Systems through Feedback Loops*. Berlin, Heidelberg: Springer-Verlag, 2009, vol. 5525/2009, pp. 48–70.

[4] J. Andersson, R. de Lemos, S. Malek, and D. Weyns, *Modeling Dimensions of Self-Adaptive Software Systems*. Berlin, Heidelberg: Springer-Verlag, 2009, vol. 5525/2009, pp. 27–47.

[5] V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos, "Awareness Requirements for Adaptive Systems," in *SEAMS '11: 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Honolulu, USA: ACM, 2011.

[6] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos, "System Identification for Adaptive Systems: a Requirements Engineering Perspective," in *ER '11: 30th International Conference on Conceptual Modeling (to appear)*, 2011.

[7] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[8] I. J. Jureta, J. Mylopoulos, and S. Faulkner, "Revisiting the Core Ontology and Problem in Requirements Engineering," in *RE '08: 16th IEEE International Requirements Engineering Conference*. Barcelona, Spain: IEEE, 2008, pp. 71–80.

[9] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

[10] V. Bryl, "Supporting the design of socio-technical systems by exploring and evaluating design alternatives," Ph.D. dissertation, Università degli Studi di Trento, Trento, Italy, March 2009.

[11] H. Garcia-Molina and K. Salem, "Sagas," in *COMAD '87: The 1987 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '87. New York, NY, USA: ACM, 1987, pp. 249–259. [Online]. Available: http://doi.acm.org/10.1145/38713.38742

[12] Y. Wang, S. A. McIlraith, Y. Yu, and J. Mylopoulos, "Monitoring and diagnosing software requirements," *Automated Software Engineering*, vol. 16, pp. 3–35, March 2009.

[13] M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard, "Reconciling System Requirements and Runtime Behavior," in *IWSSD '98: 9th international workshop on Software specification and design*, Washington, DC, USA, 1998, p. 50.

[14] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," *IEEE International Conference on Requirements Engineering*, pp. 125–134, 2010.

[15] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, "RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems," in *RE '09: 17th IEEE International Requirements Engineering Conference*. Atlanta, USA: IEEE, 2009, pp. 79–88.

[16] B. Cheng et al., "Software Engineering for Self-Adaptive Systems: A Research Roadmap," *Software Engineering for Self-Adaptive Systems*, vol. 5525/2009, pp. 1–26, 2009.

[17] *SEAMS '11: Proceeding of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. New York, NY, USA: ACM, 2011.

[18] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "An Architecture for Requirements-Driven Self-reconfiguration," in *CAiSE '09: 21st International Conference on Advanced Information Systems Engineering*, vol. 5565/2009, 2009, pp. 246–260.

[19] R. Ali, F. Dalpiaz, and P. Giorgini, "A goal-based framework for contextual requirements modeling and analysis," *Requir. Eng.*, vol. 15, pp. 439–458, November 2010. [Online]. Available: http://dx.doi.org/10.1007/s00766-010-0110-z

[20] A. Lapouchnian and J. Mylopoulos, "Modeling domain variability in requirements engineering with contexts," in *ER '09: 28th International Conference on Conceptual Modeling*. Gramado, Brazil: Springer, Nov. 2009, pp. 115–130.