Requirements-aware Systems for Self-adaptation under Uncertainty

Research Statement
Dr. Nelly Bencomo

keywords: self-adaptation, uncertainty, requirements reflection, runtime models

1 Motivation

The development of software-intensive systems is driven by their requirements. Traditional requirements engineering (RE) methods focus on resolving ambiguities in requirements and advocate specifying requirements in sufficient detail so that the implementation can be checked against them for conformance. In an ideal world, this way of thinking can be very effective. Requirements can be specified clearly, updated as necessary, and evolutions of the software design can be made with the requirements in mind.

Increasingly, however, it is not sufficient to fix requirements statically because they will change at runtime as the operating environment changes. Furthermore, as software systems become more pervasive, there is growing uncertainty about the environment and so requirements changes cannot be predicted at design-time [12, 24, 39, 1, 20]. It is considerations such as these that have led to the development of self-adaptive systems (SASs) [11], which have the ability to dynamically and autonomously reconfigure their behavior to respond to changing external conditions.

Consider a scenario involving a robot vacuum cleaner for domestic apartments. The vacuum cleaner has goals clean apartment, avoid tripping hazard and minimize energy costs. Further, it has the domain assumption energy is cheapest at night. To satisfy the avoid tripping hazard goal, a requirement is derived that it should stop operating as soon as any human activity is detected. Night operation satisfies the minimize energy costs goal. Thus all the goals are satisfiable, with no trade-offs necessary. In operation, the sound of the cleaner awakens a light sleeper. In the dark, they trip over the now stopped and therefore silent cleaner. Key features of this scenario are:

• Environmental uncertainty, in terms of resident behaviour, led to the accident;

• Disturbance of light sleepers is an emergent property;

• Tripping hazard avoidance and energy efficiency might not be so compatible after all. A switch to a day-time cleaning strategy would imply trading the energy efficiency goal off against tripping hazard avoidance;

• The night-time tripping hazard could be mitigated by deriving a new requirement that on detecting movement the cleaner stops moving but keeps the vacuum motor operating so that the sound alerts residents to its presence.

With the current state-of-the-art, the vacuum cleaner could adapt its behaviour. However any adaptation would either have to be pre-defined at design time, or would have to be a reflexive response to some monitored parameter(s), perhaps using machine-learning algorithm. An effective pre-defined response would be dependent on the requirements analyst anticipating and enumerating all environmental states and the corresponding behaviour required of the vacuum cleaner. In the case of a reflexive response, the relationship between the adaptation and the goals would be at best implicit, making verification of goal satisfaction hard or impossible.

The key argument of this research proposal is that current software engineering (SE) methods do not support well the kind of dynamic appraisal of requirements needed by an SAS and illustrated by the scenario. In most software, information about the definition and structure of requirements is lost as requirements are refined into an implementation. Even in cases where requirements monitoring is explicitly included, high-level system requirements must be manually refined into low-level runtime artefacts during the design process so that they can be monitored. The vacuum cleaner’s requirements, for example, could be monitored by instrumenting the code to sense characteristics of the environment, such as energy consumed and collision events.
Usually, however, the link between low-level sensors and the high-level goals that motivated them is not explicitly recorded in the running system. This makes it very difficult to re-assess or revise requirements at runtime when the environment changes. The only information available to the system is low-level information and so reasoning can only be carried out at this level.

This research proposal is based on a new paradigm for SE, called requirements-awareness (also known as requirements reflection), in which requirements are reified as runtime entities [7]. This would allow systems to dynamically reason about themselves at the level of the requirements - in much the same way that architectural reflection [15, 10] currently allows runtime reasoning at the level of the software architecture. We believe that requirements-awareness (i.e. requirements reflection) will support the development and management of SASs because it will raise the level of discourse at which a software system is able to reflect upon itself.

2 State-of-the-art

SASs are challenging to develop because they operate under uncertain conditions. Researchers in many fields are responding to this challenge [11]. The networking community, for example, has developed networked systems capable of autonomous changes in topology, load, tasks, and physical and logical network characteristics [18]. The intelligent agent, machine learning and planning communities have also had a long interest in autonomous and adaptive systems.

SE advances for SASs have resulted in novel software architectures and programming paradigms [11]. Novel software architectures for SASs include mechanisms for swapping out components and/or connectors at runtime (c.f., [28]). Similar techniques have been investigated in the middleware community - in particular, reflection has been used to construct middleware platforms [31, 27, 9] that allow systems to introspect about their structure at runtime, so informing their automatic reconfiguration. Well-established work on reflective middleware [10, 15] uses architecture-based models of component compositions to enable reconfiguration. Reflective architectures are now well established but only support reflection over architecture not requirements.

Important early research in requirements monitoring and diagnosis have laid the foundations for requirements reflection. Fickas and Feather [23] provide a framework for systems to monitor their executions and modify themselves at runtime to better satisfy stakeholders’ goals. Similar approaches are proposed in [21, 30, 25]. Several frameworks have been developed for the generation of software monitors from requirements models [21, 35, 33, 17, 37]. Specifically, [37] goes a step further allowing not just monitoring but also diagnosis of software requirements. However, it is our understanding the solution does not offer explicit representation of requirements and only monitors system failures and needs extensions to handle failures [37]. Recently some results have been published towards tackling this challenge [2], including ours [38].

In the research reported above, information about the definition and structure of requirements is lost as requirements are refined into implementations. Current approaches do not reify requirements as runtime objects. Rather, the requirements are used to generate other runtime artefacts, such as requirements monitors. There is therefore only a loose connection between the requirements and the resulting code. This drawback leads to fundamental limitations in how the system can converse about requirements since the system does not have access to the original requirements, but only derived artefacts that may be incomplete or at inappropriate levels of abstraction.

3 Research Challenges

In this section, we outline preliminary ideas on how to achieve the requirements-awareness vision. We present three key challenges that we see are necessary to realize requirements reflection.

**Challenge 1: Runtime representation of requirements.** The first challenge is the runtime representation of requirements in a form suitable for introspection and adaptation. Introspection implies the ability of a runtime entity to reveal information about itself. Here, adaptation refers to the ability of a program to modify entities discovered through introspection.

RE is concerned with the identification of the goals to be achieved by the system, the refinement of goals into specifications of services, and the assignment of responsibilities for services among human, physical,
and software components forming the system [36]. Goals can be refined and assigned in many different ways and a significant part of the RE process consists of exploring the alternatives and selecting the most preferable option by evaluating the impacts on the system and its organizational context. Selecting among these alternatives is critical to the success of a system, but, in a SAS, an optimal selection is notoriously hard - and perhaps impossible - to achieve before runtime due to inherent uncertainties in the environment. Runtime re-assessment of these choices is therefore crucial as a way to optimize (or satisfice) the system goals in the current context during execution.

Requirements reflection depends on a runtime representation of system requirements (i.e. its runtime model [8]) that is rich enough to support the wide range of runtime analyses outlined above concerning stakeholders’ goals, software functional and non-functional requirements, alternative choices, domain assumptions, scenarios, risks, threats, and conflicts. Such runtime representation will underpin the way a system can reason and assess requirements during runtime. To support such dynamic assessment of requirements, language features found in goal-oriented requirements modeling languages such as KAOS [36] and i* [40] hold particular promise. KAOS, for example, integrates the intentional, structural, functional, and behavioral aspects of a system, and has formal semantics permitting automated reasoning over goals.

One way to achieve a runtime representation of requirements, therefore, is to base it on goal-based RE and, in particular, to provide language support for representing, navigating and manipulating instances of a meta-model for goal modeling (e.g., the KAOS meta-model [36]). The meta-model could be provided as a set of built-in constructs to a programming language, but need not be, and, could alternatively be provided in the form of (e.g.) a library. The key point is that the meta-model provides a way to represent and maintain relationships between requirements and code that implements them. This representation must be not only readily understandable by humans but also easily manipulable by the system itself. This will allow programs to query themselves to determine requirements-relevant information, such as: What are the sub-goals of a goal? Which agents are responsible for achieving the goal? What assumptions are associated with a goal? Our preliminary research results when tackling this research challenge are shown in [38] and [6].

Figure 1: Goal and architecture synchronization.

Challenge 2: Synchronization between goals and architecture. An important purpose of requirements reflection is to enable self-adaptive systems to reason over and re-evaluate their requirements at runtime. Any re-assessments to the requirements must, of course, be reflected in the running system and the crucial link to enable this to happen is to synchronize the runtime representation of the requirements and the software architecture. We therefore see a major challenge of requirements reflection to maintain
this synchronization as either the requirements are changed from above or the architecture is changed from below.

Existing work on reflection offers a potential way to structure the runtime relationship between requirements and architecture. As an example, the reflective middleware infrastructure shown in [15] is organized into two causally-connected layers - the base-layer, which consists of the running architecture - and the meta-layer, which consists of meta-objects, accessible through a meta-object protocol (MOP), for dynamically manipulating the running architecture. We propose an analogous strategy for realizing requirements reflection: a base-layer consisting of runtime requirements objects and a meta-layer allowing dynamic manipulation of requirements objects (including stakeholders’ goals, goal refinements, alternative choices, domain assumptions, etc.). This way of structuring requirements reflection therefore leads to two strata - one for requirements and one for architecture - each comprising a causally-connected base and meta-layer. Inspired on the traditional architecture meta-model (which offers operations over components and connectors), we can define primitives for the goal-based requirements meta-model that allows the meta-level to modify the base-level for the case of the requirements stratum - e.g., add_req, delete_req, replace_req, add_goal, delete_goal, replace_goal, obtain_agent_from_goal, assign_agent_to_goal (Figure 1).

The overarching research challenge of the proposed structure in Figure 1 is to coordinate the upper requirements stratum and the lower architecture stratum. That is, there needs to be a tight semantic integration between the strata so that changes in the requirements are seamlessly effected in the architecture (and vice versa). As a simple example, if a goal is changed in the upper stratum, then the running system may identify a set of components in the architecture to replace. Put more simply, changes in the software architecture should be monitored to ensure that the requirements are not broken; changes to the requirements at runtime should be reflected in the running system by dynamic generation of changes to the software architecture.

**Challenge 3: Dealing with Uncertainty.** Representing requirements as runtime entities and synching these with architectural meta-data provide the fundamental building blocks to support dynamic re-assessment of requirements, but a key additional challenge, and one which requirements reflection is intended to help with, is to deal with the inherent uncertainties of self-adaptive systems. Uncertainties arise because of the stochastic nature of events in the environment, limited sensor capabilities, and difficulties in predicting how the modification of system services will affect agents’ behaviors and the system goals. For instance, the introduction of new capabilities into the system may produce unintended effects. For example, introducing an automatic light off switch in a house may cause residents to unconsciously use more energy in other part of the house because they feel they are already saving on lighting.

Requirements-awareness, therefore, includes a consideration of how to reason about uncertainty at runtime and how to reflect this reasoning by manipulating the requirements and architecture strata. Numerous mathematical and logical frameworks exist for reasoning about uncertainty [26]. For example, probabilistic model checkers have been used to specify and analyse properties of probabilistic transition systems [29] and Bayesian networks enable reasoning over probabilistic causal models [22]. However, only limited attention has been shown so far to the treatment of uncertainty in RE models.

Our ongoing work has the objective to develop extensions to goal-oriented requirements modeling languages to support modeling and reasoning about uncertainty in design-time and runtime models. Firstly, Dr. Bencomo was one of the authors who developed the RELAX language [39], which defines a vocabulary for specifying varying levels of uncertainty in natural language requirements and whose semantics is defined formally in terms of fuzzy branching temporal logic. Secondly, Dr. Bencomo is working on the development of a quantitative goal modelling framework that extends the goal models with a probabilistic layer for the precise specification of quality concerns using Dynamic Decision Networks (DDNs) [4] based on the work done together with her MSc student [3]. Such a framework enriches the decision-making support provided by the goal-based approach cited above and allows reasoning about partial satisfaction of softgoals (expressed with probabilities) and expected utilities [5].

We believe that uncertainty can only be handled effectively if a SAS’s requirements can be reasoned over and (e.g.) re-prioritized at runtime. This is what mandates the availability of requirements as runtime objects along with their interrelationships and dependencies, and their relationships with the architecture of the SAS with the monitorable phenomena of the environment. We also envision that requirements-awareness will allow higher-level kinds of adaptation more related to system evolution than the standard adaptation provided by standard control systems.
4 Current Results and Proposed Research

In her current project (EU Marie-Curie Project Requirements@run.time), Dr. Bencomo has been working on the design and implementation of systems with the ability to dynamically observe and reason about its requirements. The results will contribute towards the development of conceptual foundations, engineering techniques, and computing infrastructure for the access and manipulations of runtime abstractions of requirements. Currently, a prototype for the use of runtime goals has been developed [5, 38, 34, 3]. The RELAX language has been proposed to make requirements more tolerant to environmental uncertainty. Design assumptions, called Claims) [38], are applied as markers of uncertainty that document how design assumptions affect goals. Monitoring Claims at runtime has been used to drive self-adaptation [38]. By monitoring Claims during the execution of the systems, their veracity can be tested. If a Claim is falsified, the effect can be propagated to the systems goal model and an alternative (more suitable) means of goal realization will be selected, resulting in dynamic adaptation of the system to a configuration that better satisfies the goals under the prevailing environmental context.

These partial results contribute towards addressing Challenge 1 [38, 34, 5] and Challenge 2 [34, 6]. Based on the current results, we propose as future research:

1. To investigate how existing requirements languages (like RELAX) and mathematical techniques (based on Fuzzy Logic or Bayesian reasoning) to deal with uncertainty can be applied so that self-adaptive systems have run-time flexibility to temporarily suspend some requirements in favour of others that is, we envisage run-time trade-offs of requirements being made as the environment changes.

   This requires to experiment with possible different implementations of RELAX. Integration of RELAX and Claims in order to assess the validity of Claims at runtime while tolerating minor and unanticipated environmental conditions that can trigger adaptations. Initial partial results are found in [32].

2. Satisficement of non-functional requirements (NFRs) during execution is partly uncertain. Measurement of NFRs is difficult due to their vague or fuzzy nature satisfaction. NFRs may not be absolutely fulfilled yet they can be labelled as sufficiently satisfied [14]. Furthermore, NFRs usually interact among them what makes difficult to reason about their fulfillment. Probabilistic approaches to model and solve uncertainty in SASs have been seem to be a promising new research direction [11, 24, 20]. However, the use of probability in this area has been of limited study. Probability theory can be used to describe the lack of crispness about the satisfiability nature of NFRs. We argue that given a decision that requires a certain configuration, the satisfaction of a NFR can be modeled using probability distributions.

   Specifically, Dynamic Decision Networks (an extension of Bayesian Networks) offer a way to implement the trade-off between non-functional requirements as environmental conditions change. The graph-based structure of DDNs matches that of SASs. Thus, it is possible to cast the decision making of a SAS as a DDN by associating (1) decisions in the DDN with designs alternatives and (2) conditional probabilities tables with the effects of those decisions over levels of satisfaction of non-functional requirements. The required utility functions to support decision making are based on the expected utility function of a DDN. Partial results of the extension of the runtime goal-based model using Bayesian-based quantification of uncertainty are shown in [3] and [5].

3. Once software prototypes exist (from 1 and 2):

   To investigate how to relate runtime requirements decisions to changes in the software architecture during execution and in both directions. That is, there needs to be a tight semantic integration between the two layers so that changes in the requirements are seamlessly reflected in the architecture (and vice versa). Synchronization triggered by changes in the requirements (top-down changes) is easier to tackle than synchronization triggered by changes in the architecture (bottom-up changes). The latter changes may require run-time learning techniques (see Special Issue models@run.time 2012 [8]). Partial results of such a synchronization are shown in [38, 34].
5 Researcher Dr Nelly Bencomo

Dr. Bencomo has been qualified as a senior researcher with experience of at least 12 years by the European Commission when she was granted her Marie Curie project Requirements@run.time in December 2009. In the last 8 years Dr. Bencomo’s software engineering research has focused on the area of model-driven engineering and dynamically adaptive systems. During the last four years she has focused on the topic requirements engineering for adaptive systems. She has worked for different EU research projects (RUNES, DiVA, and CONNECT) and also several UK National projects. Dr. Bencomo has also been active in seeking funding at the EU level (has been a key person for 2 successful EU grants plus her own EU Marie Curie Fellowship) and 1 UK National project. During her professional activities, Dr. Bencomo has performed numerous tasks that underline her capability of independent thinking and leadership qualities. For example she is the co-founder and co-organizer of the Workshop Models@run.time in the MODELS conference (this workshop is in its 7th edition in 2012), the workshop M-ADAPT: Model-Driven Software Adaptation at ECOOP 2007, and the Requirements@run.time workshop at the Requirements Engineering Conference RE.

Dr. Bencomo has been Program Committee member of numerous international conferences and international workshops. Reviewer for different international journals and Research founding agencies (Swiss National Science Foundation, Natural Sciences and Engineering Research Council of Canada, French National Research Agency (ANR), Vienna Science and Technology Fund). Dr. Bencomo is the Program Chair of 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) in 2014.

6 Originality and innovative nature of the project

To the best of our knowledge just few initiatives with early results have been published on the topic proposed in this document. Several researchers [13, 11, 16] and upcoming conferences (as the RE Conference Series) and workshops/seminars (as the Dagstuhl seminar on Software Engineering for Self-Adaptive System in 2010, and Models@runtime in 2011) highlight the importance and the need for research in the field of systems that are able to satisfy new requirements and behaviours that were not foreseen during specification and design. Here it is worth highlighting specific relevant and recent publications that give an overview of current state-of-the-art of RE and its future directions of RE and that stress the need of requirements reflections and requirements@runtime. For instance, Cheng and Atlee offer a structured overview of RE research and delineate the scope of future research directions [11]. Ernst et al also talk extensively about the topic under the name of requirements evolution [19].

So far, researchers have not provided any concrete information on any research results or upcoming projects. The vision of systems with the ability to dynamically observe and reason about their requirements is therefore new. The proposed research topic will significantly contribute to an advance of the state of the art in the field of RE because in contrast to existing requirements specifications approaches for dynamically adaptive systems the planned approach and outcome of this project will include runtime representation of requirements (requirements@runtime) and provide tool-support.

7 Timeliness and relevance of the project

This research activity is of significance in the areas of requirements engineering, eternal software systems that are dynamically adaptive and automated software computing. Undertaking this research program will contribute to French scientific excellence in research in engineering adaptive software-intensive systems. Eternal software-intensive systems are already playing an important role in the European and international economy and it is expected their relevance will grow significantly in the coming decades. A software intensive system able to dynamically adapt become more widely used in economically critical areas such as energy, smart home energy management, transportation and other areas like e-health and public welfare. All these areas require more research efforts to develop scientific foundations, methods and tools for developing adaptive software-intensive systems that can adapt to unforeseen changes of requirements, technology, environment and operating conditions, both human and technical. Requirements specification is at the heart of such foundations, methods and tools.
References


