When the Requirements for Adaptation and High Integrity Meet (Invited Paper)

Radu Calinescu
Computer Science Research Group
Aston University
Aston Triangle, Birmingham B4 7ET, UK
R.C.Calinescu@aston.ac.uk

When it is obvious that the goals cannot be reached, don't adjust the goals, adjust the action steps.
Confucius

ABSTRACT

Two classes of software that are notoriously difficult to develop on their own are rapidly merging into one. This will affect every key service that we rely upon in modern society, yet a successful merge is unlikely to be achievable using software development techniques specific to either class.

This paper explains the growing demand for software capable of both self-adaptation and high integrity, and advocates the use of a collection of "@runtime" techniques for its development, operation and management. We summarise early research into the development of such techniques, and discuss the remaining work required to overcome the great challenge of self-adaptive high-integrity software.

Categories and Subject Descriptors
D.2.4 [Software Engineering]: Software/Program Verification—Formal methods; D.2.1 [Software Engineering]: Requirements/Specifications—Methodologies

General Terms
Performance, Reliability, Security, Verification

1. INTRODUCTION

Software is increasingly used in applications characterised by change, and is expected to adapt to variations in its environment, requirements and internal state. Over the last decade, significant effort has been dedicated to developing self-adaptive software [34] by applying the principles of autonomic computing [19, 22] to software development. This effort has delivered software capable of reconfiguring itself in response to sensor-detected changes, typically by employing a combination of heuristics, simulation and artificial intelligence techniques. While this is undoubtedly a major achievement, it is not enough for a growing range of applications in which software is bound to play a key role. Future business-, safety- and security-critical applications from areas as diverse as healthcare, transportation and finance will require self-adaptive software that is also characterised by high integrity.

Providing cost-effective healthcare to an ageing world population will require the automation of safety-critical processes associated with the monitoring and treatment of long-term conditions like diabetes and certain types of circulatory disease. Software-controlled systems integrating 24-hour patient monitoring equipment and adaptive infusion pumps1 are envisaged as potential solutions for reducing the ever increasing workflow faced by healthcare providers [23, 24]. As software-controlled infusion pumps have a poor safety record even when used on their own [35], their integration into adaptive, closed-loop control solutions raises major concerns.

In transportation, next-generation vehicles will employ safety- and security-critical self-adaptive software to respond to changes in traffic conditions, within applications that aim to improve road safety, to reduce travel time and to minimise environmental impact [16, 18]. Despite significant advances in the underlying technology, security and reliability concerns have been raised about these applications [1, 28].

In the finance industry, the gray-haired experts that were once a regular sight on stock exchange floors have long been replaced by automated trading systems. However, recent years have been characterised by an accelerated use of adaptive, business-critical software trading agents [10, 21]. Unsuitable adaptation might have been one of the causes of the still not fully explained 6th May 2010 Flash Crash that wiped $1 trillion in market value for a 20-minute period [12] and of the lower-impact but equally worrying 8.1% plunge in the natural gas price for 15 seconds on 8th June 2011 [30]. It is not the case that the software community does not know how to build high-integrity systems. On the contrary, over the last two decades academic and industrial research labs have co-developed a broad spectrum of techniques that can be used to ensure that software is both correct and robust. A by no means exhaustive list of such techniques includes formal specification, design by contract, model checking, model-driven development, and assurance arguments.

However, there is a key limitation that renders these techniques ineffective for self-adaptive software: they were devised for off-line use, typically in the design stage of the

---

1Infusion pumps are medical devices for the controlled delivery of medication and nutrients into a patient’s body.
software lifecycle. Accordingly, they operate with models, properties, assumptions and conjectures that in the case of self-adaptive software are unknown until the application is deployed and running—and which change over time. What is therefore needed for the development of software for which adaptation and high integrity requirements “meet”—termed self-adaptive high-integrity software in this paper—is a similar set of techniques that can be employed at runtime, in entirely automated ways.

This paper overviews preliminary research into the development of such ‘@runtime’ techniques, and discusses some of the remaining work required to overcome the great challenge of self-adaptive high-integrity software.

2. ‘@RUNTIME’ TECHNIQUES FOR SELF-ADAPTIVE HIGH-INTEGRITY SOFTWARE

Table 1 depicts some of the most important areas of research in which effort has been dedicated to the development of ‘@runtime’ techniques for self-adaptive high-integrity software.

Recent work by the models @ runtime research community has been exploring the effectiveness of using models of the software to guide its runtime adaptation. The “dynamic software product line” approach proposed in [31] achieves this runtime adaptation by starting with a collection of system configurations whose non-functional properties are analysed and quantified off-line. The best configuration according to a set of well-defined criteria is then decided at runtime, by using a technique termed “aspect-oriented model reasoning”. Previous projects reported in [15, 17] achieve software adaptation through the application of formal analysis to architectural models of the software involved.

One problem with using models to reason about the properties of self-adaptive software at runtime is that the changes that characterise this class of software may also affect the accuracy of the models, and thus the usefulness of their runtime analysis. This serious limitation is addressed by online learning techniques that use observations of the software behaviour to maintain the analysed models up to date. The Bayesian learning techniques proposed in [5, 13], for instance, employ this approach to calculate up-to-date estimates of the transition rates in a discrete-time Markovian model used in analysing the reliability properties of adaptive service-oriented software systems. An analogous method for predicting the response time of software components by using Kalman filter estimators is described in [36]. This method enables the use of accurate queueing models in the runtime analysis of the performance-related properties of certain types of self-adaptive software.

Quantitative model checking @ runtime has recently been introduced to improve the dependability of adaptations in autonomic software systems [7, 8, 13]. Traditional quantitative model checking represents [25] a mathematically-based technique for establishing the correctness, performance and reliability of systems that exhibit stochastic behaviour, through the off-line analysis of temporal-logic properties extended with probabilities, costs and rewards. In the ‘@runtime’ variant of the technique, this analysis is performed online, on continually updated versions of the software model and non-functional properties. The results of the analysis are used to guide adaptation in ways that guarantee that the software continues to satisfy its requirements despite changes in environment, workload and internal state. The model updates involve using the learning techniques described above to ensure that model parameters (e.g., the transition probabilities of discrete-time Markov chains or the transition rates of continuous-time Markov chains) reflect the latest changes in the software behaviour. In contrast, the updates in the analysed properties correspond to changes in the non-functional requirements of the software.

Given the potentially high overheads of quantitative model checking, using the technique successfully in a runtime setting requires the exploitation of recent research into improving its scalability [14, 26]. The research in [14] achieves significant scalability improvements by precomputing the quantitative properties of the self-adaptive software off-line, as

<table>
<thead>
<tr>
<th>@Runtime research area</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>models @ runtime</td>
<td>Models of the software are analysed at runtime, in order to choose between system configurations associated with different non-functional properties.</td>
<td>[15, 17, 31]</td>
</tr>
<tr>
<td>model learning @ runtime</td>
<td>The parameters of the models used to establish reliability and performance-related properties of self-adaptive software are estimated at runtime, based on observations of the software behaviour.</td>
<td>[5, 13, 36]</td>
</tr>
<tr>
<td>quantitative model checking @ runtime</td>
<td>Non-functional software requirements are expressed as probabilistic temporal-logic properties and analysed at runtime, to detect requirement violations and to guide adaptation.</td>
<td>[7, 8, 13, 14, 26]</td>
</tr>
<tr>
<td>runtime verification</td>
<td>Finite, partial execution traces are analysed formally to detect requirement violations, and the analysis may trigger runtime software adaptations.</td>
<td>[2, 27, 29, 32]</td>
</tr>
<tr>
<td>runtime certification</td>
<td>The dependability of self-adaptive software is certified after each runtime reconfiguration step.</td>
<td>[33]</td>
</tr>
<tr>
<td>model-driven development @ runtime</td>
<td>Runtime architectural changes are achieved through the on-line synthesis of the connectors required to include new software components into the adaptive system.</td>
<td>[3, 9, 20]</td>
</tr>
</tbody>
</table>
symbolic expressions whose parameters are the variable success and failure probabilities of the software components. The complementary approach in [26] works by restricting the runtime analysis to those parts of the model that are affected by change, and reusing the results from the previous analysis of all other parts.

Runtime verification [29, 32] is a technique that complements off-line testing with the runtime monitoring and extraction of finite software execution traces, followed by a formal analysis of these traces. The technique is particularly suitable for self-adaptive software, where the ability of off-line testing to identify requirement violations is even more limited than in the case of traditional software. In recent extensions of the technique, the runtime detection of violations in the software requirements is used to trigger adaptations that have a remedial effect [2, 27].

Runtime certification [33] refers to the on-line certification of the dependability of self-adaptive software. The technique aims to augment the fault detection, identification and reconfiguration approach from [11] with guarantees that the chosen software reconfigurations do not have a negative impact on dependability. The certification is achieved by means of model-based runtime verification.

Model-driven development @ runtime techniques were recently proposed [3, 9, 20] for the on-line synthesis of interfaces (or connectors) between the dynamically selected components of self-adaptive software systems. The approach is currently applicable to service-oriented software architectures, whose web service components expose standards-based WSDL “models” [3, 9]. These models are used to synthesise the connectors required to integrate new components into an existing software architecture as part of the adaptation process, while the framework proposed in [20] enables the formal characterisation and verification of these connectors.

3. CONCLUSION

An increasing number of safety-, security- and business-critical applications are built around self-adaptive software, as the only way to ensure that they can handle the changes intrinsic to the environments they operate in. The recent advent of cloud computing has accelerated this trend significantly through the promise of important financial gains for the applications capable of adaptively scaling up and down their resource usage without compromising high integrity.

However, the existing techniques and tools for the development of high-integrity software do not extend naturally to self-adaptive software, as they use artefacts (e.g., models and properties) that are no longer fixed when self-adaptive software is concerned. Furthermore, these techniques and tools typically require interaction with a human expert, who is expected to supply input such as models and properties to analyse, and to interpret the result of this analysis.

When the requirements for adaptation and high integrity meet, qualitatively different techniques are required. Such techniques need to be fast, low overhead, and sufficiently agile to adapt to changes in what has traditionally been considered fixed. They need to operate at runtime, with minimal or no human support. Ongoing research to devise software development techniques with these characteristics was summarised in the previous section. Key among these ‘@runtime’ techniques are on-line model learning based on observations of the self-adaptive software [5, 13, 36], and the growing collection of techniques collectively termed formal methods @ runtime [6]. More recently, the integration of several ‘@runtime’ techniques has been shown to support the development of self-adaptive high-integrity software in the area of service-based systems [4].

A lot more work is still needed to achieve effective assurances for the ever-growing spectrum of self-adaptive software that critical applications depend on, and for a larger class of reliability and performability properties. As components will increasingly join and leave complex software systems “on the fly”, the discovery of suitable software components will need to be accompanied by the discovery and learning of their behaviour and service-level agreements. New standards are required to support this approach to assembling software systems, and novel modelling and analysis techniques are needed to assess the effectiveness of new configurations when criteria including reliability, performance, cost and environmental impact are taken into account. A major challenge will be the development of scalable runtime analysis and verification techniques for self-adaptive software, using approaches that are lightweight, incremental and compositional.

Finally, new distributed protocols will be required to support the dynamic selection and monitoring of inter-component service-level agreements. These protocols will need to be able to decide combinations of service-level agreements in which individual software components may have to operate suboptimally in order to contribute to the achievement of system-level efficiency.

Acknowledgements

This work was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/H042644/1.

4. REFERENCES
