# Emerging Techniques for the Engineering of Self-Adaptive High-Integrity Software

Radu Calinescu

Department of Computer Science, University of York, UK
radu.calinescu@york.ac.uk

*The journey of a thousand miles begins beneath one's feet.*

Lao Tzu

**Abstract.** The demand for cost effectiveness and increased flexibility has driven the fast-paced adoption of software systems in areas where requirement violations may lead to financial loss or loss of life. Many of these software systems need to deliver not only high integrity but also self adaptation to the continual changes that characterise such application areas. A challenge long solved by control theory for continuous-behaviour systems was thus reopened in the realm of software systems. Software engineering needs to embark on a quest for *self-adaptive high-integrity software*. This paper explains the growing need for software capable of both self-adaptation *and* high integrity, and explores the starting point for the quest to make it a reality. We overview emerging techniques for the engineering of self-adaptive high-integrity software, propose a service-based architecture that aims to integrate these techniques, and discuss opportunities for future research.

## 1 Introduction

A growing number of software and software-controlled systems are built to adapt to changes in their environment, requirements and internal state. These *self-adaptive software systems* [19,56] can successfully reconfigure themselves in response to sensor-detected changes, typically through using a combination of heuristics, simulation and artificial intelligence techniques.

The development of successful self-adaptive software within hardly a decade since the advent of autonomic computing [35,38] is a remarkable achievement. Nevertheless, this achievement alone is insufficient for an important class of applications in which self-adaptive software plays an increasingly significant role. These are applications for which requirement violations may lead to loss of life or financial loss. Healthcare, transportation and finance are among the domains that rely on such *safety-critical* or *business-critical* applications.

Clearly, self-adaptive software used in safety-critical and business-critical applications must be characterised by *high integrity*—in the sense specified by the NIST definition [52]:

J. Cámara et al. (Eds.): Assurances for Self-Adaptive Systems, LNCS 7740, pp. 297–310, 2013.

> "*High integrity software is software that must be trusted to work dependably in some critical function, and whose failure to do so may have catastrophic results, such as serious injury, loss of life or property, business failure or breach of security.*"

This definition requires high-integrity software to "*work dependably*", which Meyer [48] equates with a combination of three properties:

1. correctness—compliance with the specification;
2. robustness—ability to withstand erroneous use outside the specification;
3. security—ability to withstand malicious use outside the specification.

As emphasised in a position paper [11] that motivated the work described here, software engineering tools for building software that is both correct and robust do exist. They include formal verification and validation (V&V), design by contract, and quality assurance [9,48,52].

However, for self-adaptive software, the three properties listed above must continue to hold as the software evolves to adapt to change. This additional requirement changes everything, because traditional software engineering approaches to developing high-integrity software were devised for off-line use during the design or V&V stages of the software lifecycle. As described in [11], "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." The range of changes that can affect self-adaptive software systems is extremely large or, in the case of large-scale complex systems like those discussed in [57], unbounded. Therefore, analysing the adaptation state space off-line is impractical in the first case, and unfeasible in the latter. Analogous techniques that can be applied automatically, while the system is running, are required for *self-adaptive high-integrity software*.

This challenge of simultaneous adaptation and high integrity has long been addressed by control theory, albeit primarily for continuous-behaviour systems (e.g., [26]). As cost savings and the need for increased flexibility have led to the replacement of these systems with software-based ones, the challenge is again open—for both software-only and embedded (or cyber-physical) systems.

The rest of the paper explores several aspects of self-adaptive high-integrity software, and discusses a service-based architecture for building software systems with these characteristics. Section 2 describes several archetypal applications that require self-adaptive high-integrity software. Section 3 overviews emerging software engineering techniques that support the development of self-adaptive high-integrity software, and discusses the current trend to implement critical software applications through the dynamic integration of heterogeneous services. Section 4 introduces a generic service-based software architecture that employs a combination of these techniques to produce *self-adaptive high-integrity software*. Finally, a preliminary research agenda is discussed in Section 5.

## 2 Critical Applications Requiring Self-Adaptive High-Integrity Software

This section overviews a selection of critical applications from three application domains, explaining why they each need software that supports both self adaptation and high integrity.

*Healthcare.* The fast ageing of the world's population is accommodated by many developed countries through healthcare budget increases that exceed the overall rate of economic growth. As this approach is unsustainable in the long term, IT-enabled *ambient assisted living* is perceived as an effective long-term solution for the monitoring of patients with chronic diseases and mobility-related conditions.

Ambient assisted living applications that employ wearable systems for health monitoring and use remote services for vital parameter analysis, medical record access, etc. could extend the time that elderly people manage independently at home, thus reducing healthcare costs and also improving their quality of life. Software-controlled systems integrating this 24-hour patient monitoring equipment with adaptive infusion pumps are envisaged as a potential extension of this solution [39,40]. (Infusion pumps are medical devices for the controlled delivery of medication and nutrients into a patient's body.) Medical conditions that could benefit from this approach include chronic cardiac and respiratory problems, diabetes, and high-risk pregnancies [33]. Nevertheless, software-controlled infusion pumps have a poor safety record even when used in a non-adaptive operating mode [58], so their integration into adaptive, closed-loop control solutions raises major concerns.

*Transportation.* In Europe alone, the transport sector is required to achieve "*a reduction of at least 60% of greenhouse gas emissions by 2050 with respect to 1990*" [22] as a contribution towards limiting climate change below 2°C. To achieve this objective, the manufaturers of next-generation vehicles and the planners of future road infrastructure will use safety–critical self-adaptive software to inform and help drivers respond to changes in traffic conditions, reducing travel time and fuel consumption, and improving road safety [30,34]. Despite significant advances in the underlying technology, security and reliability concerns have been raised about these applications [1,44].

*Finance.* In the finance industry, stock exchange transactions are increasingly carried out by automated trading systems that can react faster than their human counterparts. Furthermore, the adoption of adaptive, business-critical software trading agents in recent years has led to highly flexible applications whose effectiveness often matches that of human experts [21,37].

Nevertheless, self-adaptation in automated trading agents is a double-edged sword. 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 [24] and of the lower-impact but equally worrying 8.1% plunge in the natural gas price for 15 seconds on 8th June 2011 [49].

**Table 1.** Techniques that can help support the development and operation of self-adaptive high-integrity software

| Technique/Research area | Description | Examples |
|---|---|---|
| models @ runtime | Models of the functional and/or non-functional software behaviour are analysed at runtime, in order to select system configurations that satisfy the requirements. | [8,29,31,50] |
| on-line learning | The parameters and/or structure of the models used to establish reliability, performance or functional properties of self-adaptive software are estimated at runtime, based on observations of the software behaviour. | [7,14,25,59] |
| quantitative model checking @ runtime | Non-functional software requirements are expressed as probabilistic temporal-logic properties, and are analysed at runtime, to predict or detect requirement violations and to guide adaptation. | [16,17,25,27,42] |
| runtime verification | Finite, partial execution traces are analysed formally to detect requirements violations, and the analysis may trigger runtime software adaptations. | [6,43,45,54] |
| runtime certification | The dependability of self-adaptive software is (re)certified after each runtime reconfiguration step. | [23,55] |
| model-driven development @ runtime | Runtime architectural changes are achieved through the on-line synthesis of the connectors required to include new software components into the adaptive system. | [7,10,20,36] |

## 3 Background

### 3.1 Techniques for Self-Adaptive High-Integrity Software

This section (adapted from our previous work in [11]) describes the main research areas in which effort has been dedicated to the development of techniques that have the potential to support the realisation of self-adaptive high-integrity software. Table 1 summarises these results.

*Models @ runtime.* A growing number of research projects are investigating the use of models to steer the runtime adaptation of software systems. The types of models used by these projects range from architectural models [29,31] to parametric models of the valid system configurations [50] and data-flow automata [8].

The approach proposed in [29,31] employs formal analysis of architectural models in order to achieve software adaptation. In contrast, the "dynamic software product line" approach described in [50] achieves this runtime adaptation by starting with a collection of system configurations whose non-functional properties are analysed and quantified off-line. A technique called "aspect-oriented

model reasoning' is then used at runtime, to select and adopt the optimal configuration according to a set of well-defined requirements. Finally, the approach in [8] uses synthesised data-flow automata to model the behaviour of web services and to support their automatic composition into software applications.

*On-line learning.* The effectiveness of model-based reasoning about the properties of a software system depends on the accuracy of the models used in the analysis. This dependency is particularly relevant for self-adaptive software, where the system evolution in response to changes can easily render obsolete the very models used to guide this evolution. This serious limitation is addressed by *on-line learning* techniques that use observations of the software behaviour to maintain the analysed models up to date.

The project presented in [7], for instance, is actively working on the development of a suite of statistical and automata learning techniques for inferring the functional semantics and the behavioural semantics of networked systems, respectively.

In the related approaches proposed in [14,25], the self-adaptation of service-based systems with strict reliability requirements is achieved through the analysis of discrete-time Markovian models whose transition probabilities are learnt on-line by using Bayesian learning techniques. An analogous method for predicting the response time of software components by using Kalman filter estimators is described in [59]. 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.* Recent research aimed at improving the dependability of self-adaptive software systems has proposed the use of quantitative model checking in the runtime adaptation process [12,13,16,17,25]. Quantitative model checking [41] is a mathematically-based technique for establishing the correctness, performance and reliability of systems characterised by stochastic behaviour.

Quantitative model checking is traditionally used for the off-line analysis of system properties expressed in temporal-logics extended with probabilities, costs and rewards. In the '@runtime' variant of the technique advocated in [13,16,17,25], this analysis is performed on-line, on continually updated versions of the software model and of its 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. Maintaining the model up to date involves the application of the learning techniques described earlier in the paper [14,25,59], 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 evolution of the software behaviour. In contrast, the updates in the analysed properties correspond to user-initiated modifications 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 [27,42]. The results presented in [27] achieve significant scalability improvements by precomputing the quantitative properties of the self-adaptive software off-line, as symbolic expressions whose parameters are the variable success and failure probabilities of the software components. The complementary approach in [42] 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.* Runtime verification [45,47,54] is a technique that complements off-line testing with the runtime monitoring and extraction of finite software execution traces, followed by the analysis of these traces against a formal specification of the correct software behaviour. This specification is described using formalisms that range from temporal logics [45,54] and regular expressions [2] to state machines [4] and rule systems [5]. In extended variants of the technique, the runtime detection of violations in the software requirements is used to trigger adaptions that have a remedial effect [6,43].

Runtime verification is particularly suitable for self-adaptive software, where the ability to use off-line testing to identify requirement violations is even more limited than in the case of traditional, non-adaptive software.

*Runtime certification.* Runtime certification [55] 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 [23] 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.* Model-driven development @ runtime techniques were recently proposed [7,10,20,36] 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" [7,10,20]. 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 [36] enables the formal characterisation and verification of these connectors.

### 3.2 Critical Application Development through Service Integration

National and international strategic research agendas envisage that the types of safety-critical and business-critical applications described in Sections 1–2 will be increasingly developed through the dynamic integration of *software services* [28,51,53]. These services are expected to be flexible and shareable, to belong to multiple applications at the same time, and to self-adapt in response to change.

The research-funding programmes set up to support fundamental and applied research leading to the development of such services specify that they will need to interoperate across a range of platforms that includes private and public clouds, Internet of Things (IoT) and Internet of Contents (IoC). In other words, they need to be self-adaptive high-integrity services capable of on-the-fly integration into critical software applications that inherit the capabilities of their component services.

## 4 Towards a Service-Based Architecture Integrating "@ Runtime" Techniques for Self Adaptation and High Integrity

The vision of service-based future critical applications described above is illustrated in Figure 1, which depicts the high-level architecture of two of the applications mentioned in Section 2. The first application is an ambient assisted living system assembled from:

- wearable vital parameter monitoring (e.g., [3]) and infusion pump (e.g., [46]) IoT devices;
- medical record (MR) and vital parameter analysis (VPA) services running on a private cloud;
- public-cloud services such as emergency (EMERG), pharmacy (PHARM), accident and emergency (A&E), weather forecast (WF) and roadmap (RMAP).

The last two of these services are also part of a road traffic information system, which also comprises:

- smart-vehicle and traffic-sensor IoT components;
- private-cloud traffic analysis (TA) services.

Each legacy or newly developed component of the service-based software architecture from Figure 1 is wrapped into an appropriately configured instance of a reusable *self-adaptive high-integrity service*. This should be a standards-based service that augments traditional service-oriented architecture (SOA) functionality with enhanced versions of the techniques described in Section 3.1, therefore enabling the transparent integration of these "@runtime" techniques into critical software applications.

Figure 2 shows a possible prototype architecture for such a self-adaptive high-integrity service. The elements of this architecture employ the "@runtime" techniques from Section 3.1 as described below.

*Self-adaptive high-integrity middleware.* The application-independent, reconfigurable self-adaptive high-integrity middleware at the core of the architecture continually learns, maintains and exploits detailed, accurate and up-to-date behavioural models of peer services and of the system components it provides a wrapper for. To achieve this, the middleware uses a combination of:

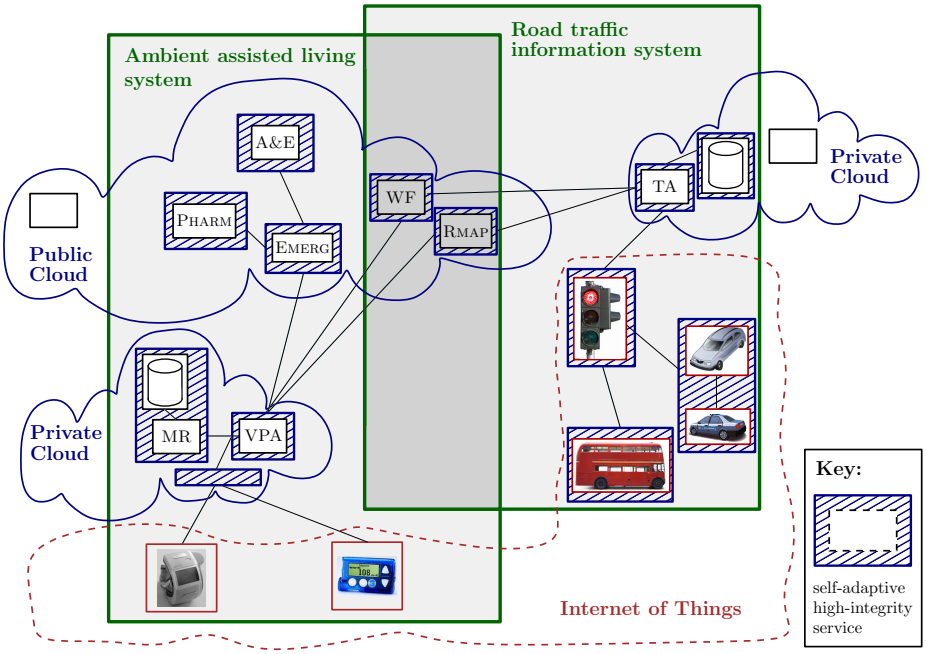


**Fig. 1.** Critical applications assembled through the cross-platform integration of interoperable services

(a) "models @ runtime" (to maintain a set of models that reflect the evolution of its own behaviour and of the behaviour of its peer services);
(b) on-line learning techniques (to update its models);
(c) quantitative model checking @ runtime, and runtime verification techniques (to guide its dependable adaptation); and
(d) runtime certification (to certify itself for the benefit of peer services and of the applications it belongs to).

*Configurator.* The domain-specific configurator is used to repurpose the self-adaptive high-integrity middleware for the application domain (or domains) that it is meant to operate in. Configured middleware will be able to "speak" the relevant domain-specific language(s) with similarly configured peer services, and with the administrators and users of applications from these domains. This will enable, for instance, the exchange and automated translation of domain-specific requirements into an internal representation that can be analysed automatically against up-to-date, on-line learnt models. The use of model-driven development @ runtime techniques will be required to synthesise the software modules supporting this functionality.

*Intelligent proxies.* The intelligent proxies linking the services belonging to the same critical application(s) represent significantly enhanced versions of the

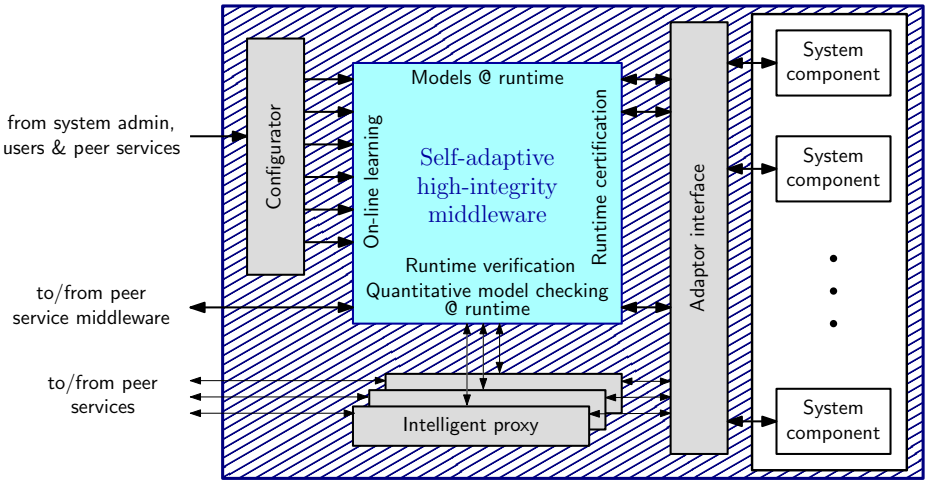


**Fig. 2.** Prototype architecture of a self-adaptive high-integrity service that uses the "@runtime" techniques from Section 3.1

traditional web service proxies used in today's SOA applications. Thus, in addition to ensuring service interoperability, the intelligent proxies will continually monitor the performance and dependability properties (e.g., response time and success rate) of peer services. They will use the data obtained from this monitoring and fast on-line learning algorithms to devise partial peer-service models that will be assembled into fully-fledged behavioural models by the self-adaptive, high-integrity middleware.

*Adaptor interfaces.* The application-specific adaptor interfaces connect heterogeneous system components (e.g., the IoT virtual objects and cloud-deployed services from the applications in Figure 1) to the middleware modules. As a result, such components:

(a) can benefit from the capabilities provided by the middleware; and
(b) can be organised into interoperable self-adaptive, high-integrity services for integration into multi-platform critical applications.

## 5 Conclusion

We argued that software engineering is unprepared for today's fast-paced adoption of self-adaptive software in safety-critical and business-critical applications. The existing approaches to engineering the high-integrity software required in such applications rely on models and properties that do not change over time, and are underpinned by high-overhead analysis techniques suited for off-line use. Neither of these prerequisites holds for self-adaptive software, which is typically developed using "best-effort" techniques that cannot guarantee requirements compliance.

Software engineering needs to embark on a quest for techniques capable of delivering high integrity and self adaptation at the same time. The outcome of this quest should include low-overhead software engineering techniques capable of fully automated, on-line operation, and novel architectures that integrate these techniques into high-integrity self-adaptive software systems.

We explored a number of emerging software engineering paradigms that collectively represent the starting point for this quest. The core principle underlying all these paradigms is that software engineering techniques traditionally used in the off-line stages of the software life cycle should be complemented by analogous techniques that are suitable for use at run time. A growing number of research projects work on new software engineering techniques that match this description. They include projects that use *models at runtime* to support the dependable evolution, formal analysis, and certification of self-adaptive software; and projects concerned with learning and updating the parameters and structure of these models continually.

Many challenges need to be overcome before we can achieve effective assurances for critical applications that use self-adaptive software. Key among these challenges is the need for runtime model analysis and verification techniques that are lightweight, incremental and compositional [15,18,32]. The ability to take full advantage of such techniques will depend on the successful development of effective approaches for the on-line learning of the analysed models.

Future software systems will comprise components that join and leave dynamically [7,57], so suitable software components will need to be discovered and their behaviour will need to be learnt "on the fly". Assembling these software systems for use in critical applications will require software architectures based on reconfigurable middleware that integrates state-of-the-art runtime learning and analysis techniques into an easy-to-use framework. We suggested a possible service-based architecture for this role and indicated how it could be built through integrating a number of emerging software engineering techniques, but significant additional work is required in this area.

Another important challenge is the development of novel approaches for achieving the levels of component interoperability required by high-integrity self-adaptive software. These approaches will have to be based on new standards for expressing a broad range of functional and non-functional properties of software components, and on scalable techniques for inferring the system-level properties from the component properties.

Future safety-critical and business-critical applications will comprise large numbers of embedded (or *cyber-physical*) systems. Ensuring that these applications achieve both high integrity and self adaptation will require the integration of the software engineering advances mentioned above with established control theory techniques.

Last but not least, there is the problem of "who watches the watchmen": the intelligent future middleware underpinning the high-integrity self-adaptive software systems of the future will itself need to be certified or self-certifiable.

**Acknowledgements.** This work was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/H042644/1. The author is grateful to Octavian Pastravanu for the insightful discussion about control theory approaches to developing continuous-behaviour self-adaptive systems.

## References

1. Aijaz, A., Bochow, B., Dotzer, F., Festag, A., Gerlach, M., Kroh, R., Leinmuller, T.: Attacks on inter vehicle communication systems - an analysis. In: Proc. 3nd Intl. Workshop Intelligent Transportation, pp. 189–194 (2006)
2. Allan, C., Avgustinov, P., Christensen, A.S., Hendren, L., Kuzins, S., Lhoták, O., de Moor, O., Sereni, D., Sittampalam, G., Tibble, J.: Adding trace matching with free variables to AspectJ. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2005), pp. 345–364. ACM (2005)
3. Anliker, U., et al.: AMON: a wearable multiparameter medical monitoring and alert system. IEEE Transactions on Information Technology in Biomedicine 8(4), 415–427 (2004)
4. Barringer, H., Havelund, K.: TraceContract: A Scala DSL for Trace Analysis. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 57–72. Springer, Heidelberg (2011)
5. Barringer, H., Havelund, K., Rydeheard, D., Groce, A.: Rule Systems for Runtime Verification: A Short Tutorial. In: Bensalem, S., Peled, D.A. (eds.) RV 2009. LNCS, vol. 5779, pp. 1–24. Springer, Heidelberg (2009)
6. Bauer, A., Leucker, M., Schallhart, C.: Model-based methods for the runtime analysis of reactive distributed systems. In: Proc. Australian Software Engineering Conference, pp. 243–252 (2006)
7. Bennaceur, A., Howar, F., Issarny, V., Johansson, R., Moschitti, A., Spalazzese, R., Steffen, B., Sykes, D.: Machine Learning for Emergent Middleware. In: Proceedings of the Joint Workshop on Intelligent Methods for Software System Engineering (2012)
8. Bertolino, A., Inverardi, P., Pelliccione, P., Tivoli, M.: Automatic synthesis of behavior protocols for composable web-services. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 141–150. ACM (2009)
9. Burton, S., Clark, J., Galloway, A., McDermid, J.: Automated V&V for high integrity systems, a targeted formal methods approach. In: NASA Langley Formal Methods Workshop (January 2000), `ftp://ftp.cs.york.ac.uk/pub/hise/NASALangley.pdf` (last retrieved on September 10, 2012)
10. Calinescu, R.: Run-time connector synthesis for autonomic systems of systems. Journal On Advances in Intelligent Systems 2(2-3), 376–386 (2009)
11. Calinescu, R.: When the requirements for adaptation and high integrity meet. In: Proceedings of the 8th Workshop on Assurances for Self-Adaptive Systems (ASAS 2011), pp. 1–4. ACM, New York (2011)
12. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimisation in service-based systems. IEEE Transactions on Software Engineering 37(3), 387–409 (2011)

13. Calinescu, R., Ghezzi, C., Kwiatkowska, M., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Communications of the ACM 55(9), 69–77 (2012)
14. Calinescu, R., Johnson, K., Rafiq, Y.: Using observation ageing to improve Markovian model learning in QoS engineering. In: Proceedings 2nd ACM/SPEC International Conference on Performance Engineering, pp. 505–510 (2011)
15. Calinescu, R., Kikuchi, S.: Formal Methods @ Runtime. In: Calinescu, R., Jackson, E. (eds.) Monterey Workshop 2010. LNCS, vol. 6662, pp. 122–135. Springer, Heidelberg (2011)
16. Calinescu, R., Kwiatkowska, M.: CADS*: Computer-Aided Development of Self-* Systems. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 421–424. Springer, Heidelberg (2009)
17. Calinescu, R., Kwiatkowska, M.: Using quantitative analysis to implement autonomic IT systems. In: Proceedings of the 31st International Conference on Software Engineering (ICSE 2009), pp. 100–110 (2009)
18. Calinescu, R., Kikuchi, S., Johnson, K.: Using Compositional Verification to Manage Change in Large-Scale Complex IT Systems. In: Large-Scale Complex IT Systems - Development, Operation and Management. LNCS, vol. 7539, pp. 303–329. Springer (2012)
19. Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H.M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H.A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J.: Software Engineering for Self-Adaptive Systems: A Research Roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)
20. Cavallaro, L., Di Nitto, E., Pelliccione, P., Pradella, M., Tivoli, M.: Synthesizing adapters for conversational web-services from their WSDL interface. In: ICSE 2010 SEAMS: Workshop on Software Engineering for Adaptive and Self-Managing Systems, pp. 104–113 (2010)
21. Collins, J., Ketter, W., Gini, M.: Flexible decision control in an autonomous trading agent. Electronic Commerce Research & Appl. 8(2), 91–105 (2009)
22. COM(2011) 144: European Commission. Roadmap to a Single European Transport Area Towards a competitive and resource efficient transport system (2011), http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2011:0144:FIN:EN:PDF (last retrieved on September 10, 2012)
23. Crow, J., Rushby, J.: Model-based reconfiguration: Diagnosis and recovery. NASA Contractor Report 4596, NASA Langley Research Center, Hampton, VA (Work performed by SRI International) (May 1994)
24. Easley, D., de Prado, M.M.L., O'Hara, M.: The microstructure of the 'Flash Crash': Flow toxicity, liquidity crashes and the probability of informed trading. Journal of Portofolio Management 37(2), 118–128 (2011)
25. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time adaptation. In: Proceedings of the 31st International Conference on Software Engineering, pp. 111–121. IEEE Computer Society Press (2009)
26. Feng, G., Lozano, R.: Adaptive Control Systems. Elsevier (1999)
27. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: Proceedings of the 33rd International Conference on Software Engineering, IEEE Computer Society (2011)

28. Future Internet Assembly. Research Roadmap Towards Framework 8: Research Priorities for the Future Internet (2011), http://fisa.future-internet.eu/images/0/0c/Future_Internet_Assembly_Research_Roadmap_V1.pdf
29. Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E.: Using architecture models for runtime adaptability. IEEE Software 23, 62–70 (2006)
30. Fritsch, S., Senart, A., Schmidt, D.C., Clarke, S.: Time-bounded adaptation for automotive system software. In: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008, pp. 571–580. ACM, New York (2008)
31. Garlan, D., Schmerl, B.R.: Using architectural models at runtime: Research challenges. In: European Workshop Software Architecture, pp. 200–205 (2004)
32. Ghezzi, C.: Evolution, adaptation and the quest for incrementality. In: Preproceedings of the 17th Monterey Workshop on Development, Operation and Management of Large-Scale Complex IT Systems, pp. 79–88 (2012)
33. Ghini, V., Ferretti, S., Panzieri, F.: M-Hippocrates: Enabling Reliable and Interactive Mobile Health Services. IT Professional 14(3), 29–35 (2012)
34. Hartenstein, H., Laberteaux, K.P. (eds.): VANET: Vehicular Applications and Inter-Networking Technologies. John Wiley & Sons (2009)
35. Huebscher, M.C., McCann, J.A.: A survey of autonomic computing—degrees, models, and applications. ACM Comp. Surveys 40(3), 1–28 (2008)
36. Issarny, V., Bennaceur, A., Bromberg, Y.-D.: Middleware-Layer Connector Synthesis: Beyond State of the Art in Middleware Interoperability. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 217–255. Springer, Heidelberg (2011)
37. Izumi, K., Toriumi, F., Matsui, H.: Evaluation of automated-trading strategies using an artificial market. Neurocomputing 72(16-18), 3469–3476 (2009)
38. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer Journal 36(1), 41–50 (2003)
39. Kovatchev, B.: Closed loop control for type 1 diabetes. British Medical Journal 342, d1911 (2011)
40. Kramer, G.C., Kinsky, M.P., Prough, D.S., Salinas, J., Sondeen, J.L., Hazel-Scerbo, M.L., Mitchell, C.E.: Closed-loop control of fluid therapy for treatment of hypovolemia. Journal of Trauma-Injury Infection & Critical Care 64(4), S333–S341 (2008)
41. Kwiatkowska, M.: Quantitative verification: Models, techniques and tools. In: Proc. 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. Foundations of Software Engineering, pp. 449–458. ACM Press (2007)
42. Kwiatkowska, M., Parker, D., Qu, H.: Incremental quantitative verification for Markov decision processes. In: Proceedings 2011 IEEE/IFIP International Conference Dependable Systems and Networks (2011)
43. Kyas, M., Prisacariu, C., Schneider, G.: Run-Time Monitoring of Electronic Contracts. In: Cha, S(S.), Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 397–407. Springer, Heidelberg (2008)
44. Lee, U., Cheung, R., Gerla, M.: Emerging vehicular applications. In: Olariu, S., Weigle, M.C. (eds.) Vehicular Networks: From Theory to Practice. Chapman and Hall/CRC (2009)
45. Leucker, M., Schallhart, C.: A brief account of runtime verification. Journal of Logic and Algebraic Programming 78(5), 293–303 (2009)

46. Mastrototaro, J., Lee, S.: The Integrated MiniMed Paradigm Real-Time Insulin Pump and Glucose Monitoring System: Implications for Improved Patient Outcomes. Diabetes Technology & Therapeutics 11(s1), 37–43 (2009)
47. Meredith, P., Roşu, G.: Runtime Verification with the RV System. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) RV 2010. LNCS, vol. 6418, pp. 136–152. Springer, Heidelberg (2010)
48. Meyer, B.: Dependable Software. In: Kohlas, J., Meyer, B., Schiper, A. (eds.) Dependable Systems: Software, Computing, Networks. LNCS, vol. 4028, pp. 1–33. Springer, Heidelberg (2006)
49. Meyer, G.: Traders flummoxed by natural gas 'flash crash'. Financial Times (June 9, 2011)
50. Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F., Solberg, A.: Models@run.time to support dynamic adaptation. Computer 42, 44–51 (2009)
51. Networked European Software and Services Initiative. Research Priorities for the next Framework Programme for Research and Technological Development FP8 (May 2011), http://www.nessi-europe.com/files/Docs/NESSI%20SRA_update_May_2011_V1-0.pdf
52. Wallace, D.R., Ippolito, L.M., Kuhn, D.R.: High Integrity Software Standards and Guidelines. NIST SP 500-204, National Institute of Standards and Technology, Gaithersburg, MD, 20899 (September 1992)
53. National Science Foundation. Cyberinfrastructure Framework for 21st Century Science and Engineering. A Vision and Strategy for Data in Science, Engineering, and Education (April 2012), http://www.nsf.gov/od/oci/cif21/DataVision2012.pdf
54. Pnueli, A., Zaks, A.: PSL Model Checking and Run-Time Verification Via Testers. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 573–586. Springer, Heidelberg (2006)
55. Rushby, J.: Runtime Certification. In: Leucker, M. (ed.) RV 2008. LNCS, vol. 5289, pp. 21–35. Springer, Heidelberg (2008)
56. Salehie, M., Tahvildari, L.: Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst. 4(2), 1–42 (2009)
57. Sommerville, I., Cliff, D., Calinescu, R., Keen, J., Kelly, J.T., Kwiatkowska, M., McDermid, J., Paige, R.: Large-scale complex IT systems. Communications of the ACM 55(7), 71–77 (2012)
58. Food, U.S.: Drug Administration — Center for Devices and Radiological Health. Infusion pump improvement initiative, White paper (April 2010), http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/GeneralHospitalDevicesandSupplies/InfusionPumps/ucm205424.htm (last retrieved on September 10, 2012)
59. Zheng, T., Woodside, M., Litoiu, M.: Performance model estimation and tracking using optimal filters. IEEE Transactions on Software Engineering 34(3), 391–406 (2008)