A Survey of Formal Methods in Self-Adaptive Systems

Danny Weyns, M. Usman Iftikhar, Didac Gil de la Iglesia, Tanvir Ahmad
Department of Computer Science
Linnaeus University
Växjö, Sweden
danny.weyns@lnu.se

ABSTRACT

One major challenge in self-adaptive systems is to assure the required quality properties. Formal methods provide the means to rigorously specify and reason about the behaviors of self-adaptive systems, both at design time and runtime. To the best of our knowledge, no systematic study has been performed on the use of formal methods in self-adaptive systems. As a result, there is no clear view on what methods have been used to verify self-adaptive systems, and what support these methods offer to software developers. As such insight is important for researchers and engineers, we performed a systematic literature review covering 12 main software engineering venues and 4 journals, resulting in 75 papers used for data collection. The study shows that the attention for self-adaptive software systems is gradually increasing, but the number of studies that employ formal methods remains low. The main focus of formalization is on modeling and reasoning. Model checking and theorem proving have gained limited attention. The main concerns of interest in formalization of self-adaptation are efficiency/performance and reliability. Important adaptation concerns, such as security and scalability, are hardly considered. To verify the concerns of interest, a set of new properties are defined, such as interference freedom, responsiveness, mismatch, and loss-tolerance. A relevant part of the studies use formal methods at runtime, but the use is limited to modeling and analysis. Formal methods can be applied to other runtime activities of self-adaptation, and there is a need for light-weight tools to support runtime verification.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architecture; D.2.4 [**Software Engineering**]: Software/Program Verification—*formal methods*

General Terms

Theory, verification, documentation

Keywords

Self-adaptive systems, systematic literature review

C3S2E-12 June 27-29, 2012 Montreal, QC, CANADA.
Editors: B. C. Desai, S. Mudur, E. Vassev

1. INTRODUCTION

Self-adaptation has been widely recognized as an effective approach to deal with the increasing complexity and dynamicity of modern software systems [67, 53, 58]. A self-adaptive system comprises two parts: the managed system (also called system layer [42], managed resources [53], core function [71], base-level subsystem [91]) that deals with the domain functionality, and the managing system (or architecture layer [42], autonomic manager [53], adaptation engine [71], reflective subsystem [91]) that deals with the adaptations of the managed system to achieve particular quality objectives. The key underlying idea of self-adaptation is complexity management through separation of concerns. One major challenge in self-adaptive systems is to provide guarantees about the required quality properties. Formal methods provide the means to rigorously specify and verify the behavior of self-adaptive systems. Formal methods have been applied during system development, but also during runtime to provide guarantees about the required properties of self-adaptive systems [61, 94, 83, 91, 81].

In 2004, Bradbury et al. [15] surveyed 14 formal specification approaches for self-adaption based on graphs, process algebras, and logic formalisms. The survey evaluated the ability of each approach to specify self-managing systems, and concluded that existing approaches need to be enhanced to address issues regarding expressiveness and scalability. [35, 50, 71, 31, 34] summarize achievements of the field and outline challenges for future work, including challenges with respect to the application of formal methods for verification and validation of self-adaptive systems.

These studies provide insight in the potential use of formal methods in self-adaptive systems. However, to the best of our knowledge, no systematic study has been performed on the actual use of formal methods in self-adaptive systems. As a result, there is no clear view on what methods have been used to specify and verify self-adaptive systems, and what support these methods actually offer to software developers. However, such an insight is important for researchers and engineers.

The overall objective of this paper is to identify which formal methods have been used in self-adaptive systems and for what purpose these methods have been used. We also aim to assess for what type of self-adaptive applications formal methods have been applied, and which tools authors have used. To that end, we have performed a systematic literature review. From the study, we derive conclusions concerning state of the art formal methods in self-adaptive systems, and suggest ideas for future research. All the material that was used for the study together with the extracted data is available at *http://homepage.lnu.se/staff/daweaa/SLR-FMSAS.htm*.

Paper Overview. In section 2 we give an overview of the method we used in our study. In section 3 we explain study planning. We explain the research questions we address, define search strategy

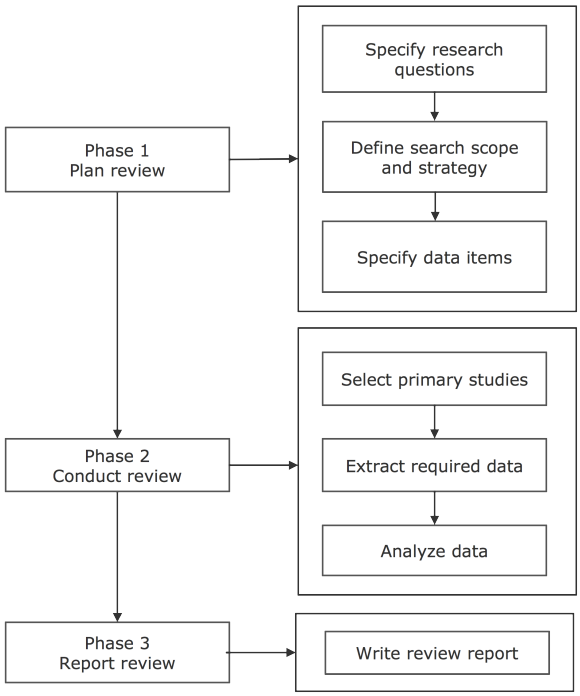

Figure 1: Overview of the systematic review process (adapted from [17]).

and scope, and summarize the data items that were collected. In section 4 we present the data extracted from the primary studies (i.e., the selected studies after filtering), and interpret this data answering the research questions. Section 5 discusses limitations of our study. We conclude with a discussion of opportunities for future research in section 6.

2. RESEARCH METHOD

Our study followed the principles of a systematic literature review [55], which is a well-defined approach to identify, evaluate and interpret all relevant studies regarding a particular research question, topic area or phenomenon of interest. Figure 1 shows an overview of the three-phased method we applied in the study.

Four researchers were involved in the literature study. In review planning (Phase 1), the review protocol was defined, which includes the definition of research questions, the search strategy and scope, and the data items that had to be collected. The protocol was defined interactively by the team of reviewers. The research questions express the research topics of interest in this literature review. The scope of the review was based on the identification of the main workshops, conferences, and journals in the field. As search strategy, we combined automatic with manual search. Automatic search was defined as a two-step process for which two search strings were defined. The first string aims to select the studies on self-adaptive systems, and the second string aims to filter the studies on formal methods. For the manual search, inclusion and exclusion criteria were defined. Next, data items were identified and for each item a set of options was defined. The definition of data items was based on information derived from literature sources and from experiences with a preceding literature review [89]. For some of the data items, additional options were introduced during the review process, in particular for the set of languages used for modeling systems and specifying properties, and the set of tools. The protocol was crosschecked by an external reviewer and the feedback was used to make small adaptations.

Subsequently, the four researchers conducted the review (Phase 2). Studies were automatically selected based on the search criteria defined in Phase 1. One reviewer was responsible for automatic search. Manual search was performed by two researchers that checked each paper independently based on inclusion/exclusion criteria, to select the studies for answering the research questions of the study. Conflicts were resolved and if no consensus could be reached, the other researchers were involved to come to a decision. Once the primary studies were selected, the data items were collected by the four reviewers, which was obviously a manual process. Collected data items were crosschecked and in case of disagreement, conflicts were resolved. Finally, the data derived from the the primary studies was collated and summarized to answer the research questions.

One of the reviewers coordinated the writing of this review report (Phase 3), in close consultation with the other reviewers.

3. PLANNING REVIEW

During planning, the protocol for the review is defined, which includes three main steps: specify research questions, define search strategy and scope, and define data items.

3.1 Research Questions

We formulated the general goal of the study through Goal-Question-Metric (GQM) perspectives (purpose, issue, object, viewpoint) [8]:

Purpose: Understand and characterize

Issue: the use of formal methods

Object: in self-adaptive software systems

Viewpoint: from a researcher's and engineer's viewpoint.

The general research question translates to four concrete research questions:

RQ0: Which are the trends in applying formal methods in self-adaptive systems?

RQ1: Which formal methods have been used in self-adaptive systems?

RQ2: For which adaptation concerns have formal methods been used?

RQ3: How have formal methods been used to deal with concerns of self-adaptive systems?

With RQ0, we want to get insight in the use of formal methods by researchers on self-adaptive systems both in time and space. RQ1 is motivated by the need to get insight in *what* kind of formal methods have been used for self-adaptive systems. This question aims to assess which languages have been used for modeling systems, verifying properties, and which tools have been used for this. With RQ2, we aim to understand *why* formal methods have been used in self-adaptive systems. Concretely, we aim to assess for which concerns formal methods have been used (reliability, performance, functional correctness of adaptations, etc.), which properties have been verified for that (safety, liveness, etc.), and for which type of applications these methods have been used. Finally, RQ3 aims to access *how* formal methods have been used to provide guarantees about concerns in self-adaptive systems (reasoning, model checking, at design time, runtime, etc.).

3.2 Search Scope and Strategy

The scope of the search is defined in two dimensions: time and space. Regarding time, we searched studies published from Jan. 2000 until Dec. 2011. 2000 was used as starting year since self-adaptive systems became subject of active research around that time. Regarding space, we included the primary venues for publishing research results on self-adaptive systems, as well as the major conferences and journals on software engineering. The selected venues are listed in Table 1. The Rank is based on the Australian Research Council ranking (www.arc.gov.au/era/era_2010/).

Table 1: Searched conferences and journals

ID	Venue	Rank
ASE	International Conference on Automated Software Engineering	A
Adaptive	Adaptive and Self-adaptive Systems and Applications	n/a
DEAS	Design and Evolution of Autonomic Application Software	n/a
FSE	Foundations of Software Engineering	A
ICAC	International Conference on Autonomic Computing	B
ICSE	International Conference on Software Engineering	A
ICSM	International Conference on Software Maintenance	A
ISSTA	International Symposium on Software Testing and Analysis	A
SASO	Self-Adaptive and Self-Organizing Systems	n/a
SEAMS	Software Engineering for Adaptive and Self-Managing Systems	n/a
WICSA	Working International Conference on Software Architecture	A
WOSS	Workshop on Self-Healing	n/a
JSS	Journal of Systems and Software	A
TAAS	Transactions on Autonomous and Adaptive Systems	n/a
TOSEM	Transactions on Software Engineering and Methodology	A*
TSE	Transactions on Software Engineering	A*

Our search strategy combines automatic with manual search. Automatic search comprised of two steps: first we selected the studies that are relevant for self-adaptive systems, and then we filtered the studies that use formal methods. We used the following search string in the first step:

> ((Title:adaptive OR Title:adaptation OR Title:self OR Title:autonomic OR Title:autonomous) OR (Abstract:adaptive OR Abstract:adaptation OR Abstract:self OR Abstract:autonomic OR Abstract:autonomous))

The keywords provide the main terms that different communities use to refer to self-adaptive systems. We applied pilot searches on the set of studies from SEAMS, ICAC and TAAS to ensure that the keywords provide the right scope.

In the second step, we used the following search string:

> "Transition system" OR automata OR Markov OR Petri OR "state machine" OR calculus OR algebra OR grammar OR CSP OR "temporal logic" OR LTL OR "tree logic" OR CTL OR "first order logic" OR "reaction rule" OR probabilistic OR stochastic OR "graph rewriting" OR "graph representation"

This set of keywords defines key terms that refer to the use of formal methods. We derived the terms from four sources: Baier and Katoen's book on model checking [4], the survey of Bradbury et al. on formal methods in self-adaptive systems [15], Villegas' paper in quality evaluation of self-adaptive systems [84], and Tamura et al.'s recent roadmap paper on verification of self-adaptive systems [81]. For the second filtering, we again applied pilot searches to ensure that the keywords provide the right scope of studies.

We further refined the studies resulting from automatic search using a manual search step. The goal of this step is to identify the primary studies that are directly related to the research questions. To that end, we defined the following inclusion/exclusion criteria:

- *Inclusion criterion 1:* The study formalizes (at least a part of) the system as well as properties of the system. Rationale: we might have included studies which employ some formal terms, but do not actually employ formal methods for a particular purpose related to self-adaptation.
- *Inclusion criterion 2:* The study separates the domain logic and the adaptation logic, that is, the mechanisms to realize adaptation are not (or only partially) interwoven with the regular functionality. Rationale: the focus of our study is on self-adaptive systems that separate the adaptation concerns from the domain functionality.
- *Exclusion criterion 1:* A study that is an editorial, abstract or a short paper. Rationale: these studies do not provide a reasonable amount of information.
- *Exclusion criterion 2:* A study that focuses on formal techniques themselves, rather than on the use of formal methods for self-adaptation concerns. Rationale: these studies do not provide information regarding the research questions.

A study was selected when it met both inclusion criteria and eliminated if it met any of the exclusion criterion.

3.3 Data Items

Items were defined to collect the necessary data from the studies to answer the research questions. For each primary study, the data items shown in Table 2 were collected. Each study was read in detail by two reviewers, each reviewer extracting the data in a form. This data were used during a discussion to resolve conflicts and reach consensus for the data items. This discussion involved the other reviewers when no agreement could be reached. Note that the lists of options of some of the data items were extended during the review.

The data items author(s), and title (F1-F2) were used for documentation. Year (F3) and Venue (F4) are used for answering RQ0.

For modeling language (F5) we defined the following options: regular algebra (basic math, equations, set theory, etc.), state machines, transition systems, automata, Markov models (inc. Markov chains, Markov decision processes, etc.), process calculi (CSP, π-calculus, etc.), Petri nets, graphs, Z notation, ADL (formally founded architecture description languages). This list was extended while the data was collected.

Table 2: Data collection form

Item	Field	Research question
F1	Author(s)	Documentation
F2	Title	Documentation
F3	Year	RQ0
F4	Venue	RQ0
F5	Modeling language	RQ1
F6	Property specification/verification	RQ1
F7	Concerns subject of formal verification	RQ2
F8	Verified properties	RQ3
F9	Use of formalization	RQ3
F10	Offline vs. runtime use of formal methods	RQ3
F11	Tools used for modeling and verification	RQ1
F12	Types of software systems	RQ2

Property specification/verification (F6) options are: modeling language (the same language is used for modeling the system and specifying properties), first-order logic (FOL), linear temporal logic (LTL), computation tree logic (CTL), timed computation tree logic (TCTL), probabilistic computation tree logic (PCTL), and graph grammar. This initial list was extended while data was collected from the studies.

Concerns subject of formal verification (F7) refers to the self-adaptive aspects of interest of formalization. The options (based on IEEE 9126 and ISO/IEC 25012) are:

- reliability (fault tolerance, recoverability): capability of software to maintain its level of performance under stated conditions for a stated period of time
- availability: the degree to which the software is in a functioning condition, i.e. capable to perform its intended functions
- usability (ease of learning, communicativeness): effort needed to use the system
- efficiency/performance (time behavior, resource utilization): efficiency of the software by using the appropriate resources under stated conditions and in a specific context of use
- maintainability (analyzability, changeability, stability, testability): effort needed to make specified modifications
- portability: ability of software to be transferred from one environment to another
- security: ability of the system to protect against misuse
- accuracy: the extent to which the software realizes the intended behavior in a specific context of use
- flexibility in use: capability of the software to provide quality in the widest range of contexts of use, incl. dealing with unanticipated change and uncertainty
- functionality of the self-adaptive system
- other.

Verified properties (F8) refers to the concrete properties that are subject of reasoning or verification. The options are safety ("nothing bad should happen"), liveness ("something good will happen eventually"), deadlock (components sharing the same resource are effectively preventing each other from accessing the resource), reachability (there is a finite path for reaching a certain set of states), stability (the behavior stays within well defined boundaries), and functional correctness (the system behavior complies to a specification). Besides these traditional properties for verification, we extended the list with additional properties during data collection as defined by the authors of the studies.

The options for use of formalization (F9) are: modeling, reasoning, model checking, proving, and other. Options for the time when formal methods are used (F10) are: offline (formal methods are applied during development or maintenance activities) and runtime (formal methods are used by the system itself at runtime). For tools used for formal modeling and verification (F11), we started from the following set of popular tools SPIN (general tool for verifying the correctness of distributed software models, spinroot.com/spin), PRISM (probabilistic symbolic model checker, www.prismmodelchecker.org), Uppaal (integrated tool environment for real-time systems modeled as networks of timed automata, www.uppaal.org), LTSA (tool for analyzing labelled transition systems, www.doc.ic.ac.uk/ltsa), CZT (tools for formal modeling in the Z, czt.sourceforge.net). During data collection, this list of options was further extended.

Finally, based on a sample of the studies we identified the following options for types of software systems (F12) for which self-adaption is used: parallel computing (grid, parallel computing, cloud computing, etc.), service-based systems (webservices, business applications, e-commerce, etc.), client-server systems (web applications, application server systems, etc.), embedded systems (automotive systems, mobile applications, robotic systems, etc.) and other.

4. RESULTS

We now give an overview of the study results based on the research questions we defined for our study.

RQ0: Which are the trends in applying formal methods to self-adaptive systems?

Trends are derived from the year (F3) and venue (F4). From the 6353 studies in total, 1027 were selected after applying the first search string (i.e., potential studies related to self-adaptation). Applying the second search string resulted in 489 studies (i.e., potential studies related for formal methods). From these, 75 studies were selected for final review during manual search. The list of selected studies is added in Appendix A.

Fig. 2 shows the number of studies on formal methods in self-adaptation over time. While the absolute numbers are remarkably low, there is a clear progression in the number of studies that use formal methods. 2005 is clearly a pivotal year, which is obviously connected with the creation of ICAC, SEAMS, and TAAS around that time.

We also looked at relative numbers. Between 2000 and 2005 on average 6% of the total number of studies focused on self-adaptation, while this number increased to 17% in the period 2006 - 2011. Within the studies that focus on self-adaptation, between 2000 and 2005, on average, 3% use formal methods, while this number increased to 8% between 2006 and 2011.

Fig. 3 shows the number of studies on formal methods in self-adaptation per venue. SEAMS represents 20.0% of the studies, ICAC 12.0%, and SASO 6.6%. 14.7% of the studies are published

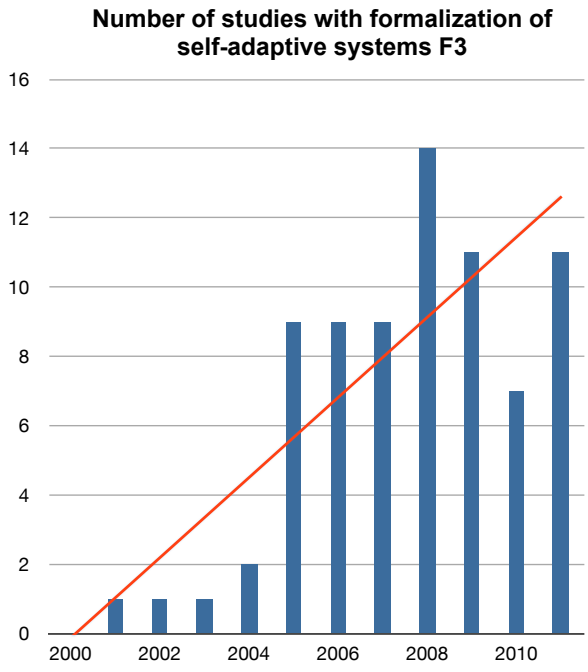

Figure 2: Overview of the studies on formal methods in self-adaptation over time

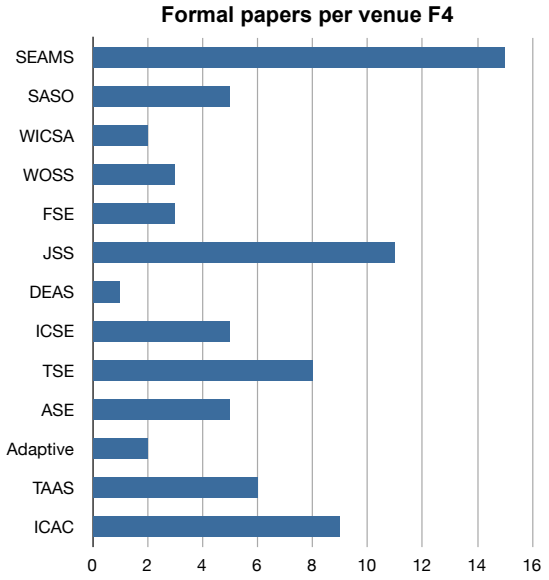

Figure 3: Overview of the studies on formal methods in self-adaptation per venue

in JSS, 10.7% in TSE and 8.0% in TAAS. Note that none of studies on self-adaptive systems of ISSTA, ICSM and TOSEM use formal methods (from 4, 19, and 8 studies on self-adaptation respectively). The top software engineering conferences FSE, ICSE, and ASE represent 17.3% of the studies.

In summary, we notice a good coverage of research results from different conferences and journals as well as increasing attention for the use of formal methods in self-adaptive systems over time. Nevertheless, the absolute numbers give a strong indication that there is a dearth of approaches that employ formal methods for assuring that self-adaptive software systems satisfy user requirements and meet their expected quality attributes. [81] argues that adoption of self-adaptation by industry is still limited due to a lack of validation and verification methods. The results of our study confirm that research and development in assurance techniques is a necessity to unlock the potential of self-adaptation beyond academic settings.

RQ1: Which methods have been used in self-adaptive systems?

To answer this question, we used data extracted from modeling languages (F5), property specification/verification (F6), and tools used for modeling and verification (F11).

Fig. 4 shows the distribution of the modeling languages used for formalization.

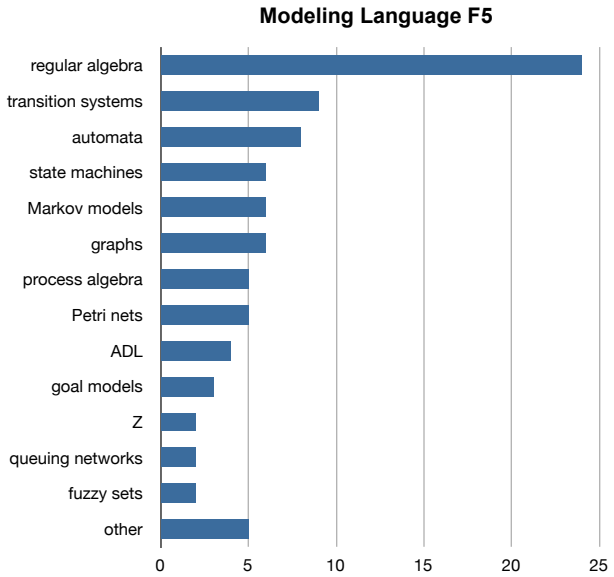

Figure 4: Modeling languages used for formalization

The main observation is that regular algebra is by far the most popular modeling language (27.6% of the studies). The use of traditional formal modeling languages such as transistion systems, automata, state machines, Markov models, graphs, and process algebras is about equally distributed (each representing 6 to 10%). We could not identify any trend in the use of the different modeling languages over time.

Fig. 5 shows the distribution of the property specification languages used for formalization.

The majority of the studies (42.7%) formulate the properties of interest in the modeling language they use for modeling. In particular, 87.5% of the studies that use algebra as modeling language use the same language for property specification. 36.0% of the studies use some logic as property specification language. Logic is combined with different kind of modeling languages.

From the data derived from data item F11, we found that 40.0% of the studies use tools for formal modeling or verification. 30.0% of the studies that use tools employ it for model checking (actually all studies that do model checking use a tool). The remaining studies that use tools employ them for modeling systems, and only one study uses a tool for proving. The most used tools are PRISM (3 studies), SPIN, LTSA, CZT, and ACME studio (each 2 studies).

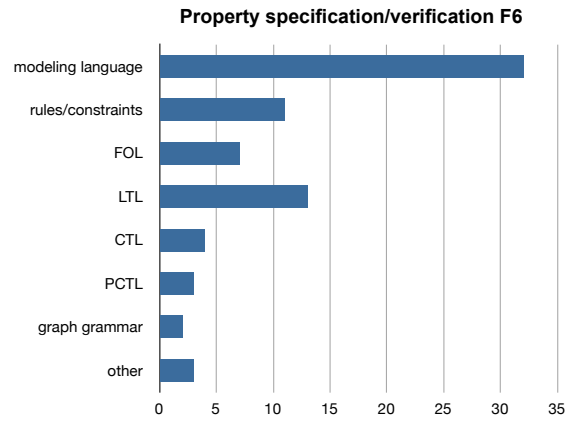

Figure 5: Property specification languages used for formalization

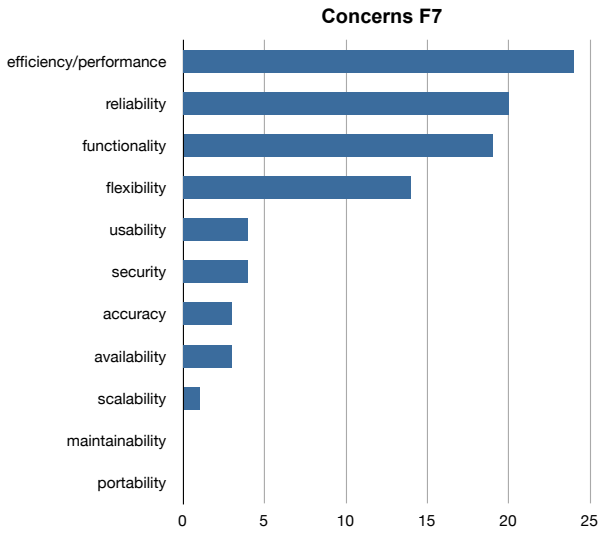

Figure 6: Adaptation concerns

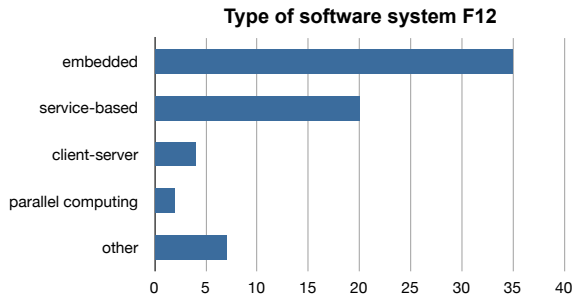

Figure 7: Types of software systems used for formalization

However, no standard tools have been emerged for formal modeling and verification of self-adaptive systems.

In summary, we notice that the majority of researchers employ regular algebraic notations both for modeling and property specification of self-adaptive systems. The formal notations are mostly used to provide a level of rigor in the explanations and discussion. However, the use of formal methods for providing evidence of system properties, e.g., by using model checking or theorem proving, remains limited. We further notice that only a few researchers make their models and study results publicly available. This indicates that current research efforts on formal methods in self-adaptive systems are not well integrated. Finally, we observe that a number of authors reuse tools that are developed for offline analysis to perform runtime analysis; a typical example is [21]. While this is surely valid in the context of an academic case study, such an approach is often not appropriate from a practical point of view. This observation indicates that there is a need for light-weight pluggable tools that support formal verification at runtime.

RQ2: For which adaptation concerns have formal methods been used?

To answer this question, we drew on data extracted from concerns subject of formalization (F7), and application domains (F12).

Fig. 6 shows the data derived for concerns of formalization.

The distribution of the concerns of self-adaptation in this study confirms the distribution we found in a previous study [89] (the scope of that study was limited to SEAMS). Top concerns of self-adaptation for which formalization is used are efficiency/performance (32.0%), reliability (26.7%), guaranteeing functionality (22.7%), and flexibility (18.7%).

Fig. 7 shows the distribution of types of software systems for which formal methods have been used.

About half of the software systems (46.7%) for which formal methods have been used are embedded systems, followed by service-based systems (26.7%).

The use of formal methods for self-adaptation in embedded systems relates to requirements on resource constraints and real time behavior of these systems. Service-based systems have gained an increasing attention during the last five years. In particular dynamic composition of services is an active area of research in self-adaptive system, with particular attention for resolving behavioral mismatches between services and dynamic selection of services to guarantee service level agreements. Nevertheless, the data extracted for RQ2 shows that formal methods are hardly considered for guaranteeing important concerns of modern software systems such as security and scalability. Furthermore, data intensive domains, such as telecommunication, and scientific domains such as climate research and bioinformatics, have gained little attention. These concerns and domains offer areas for future research on formal methods and self-adaptation.

RQ3: How have formal methods been used to deal with concerns of self-adaptive systems?

To answer this question, we used data from verified properties (F8), type of verification (F9), and offline vs. runtime use of formal methods (F10).

Fig. 8 shows the distribution of the properties that have been verified in the studies.

Traditional properties, including safety (18.7%), liveness and reachability (each 10.7%), and deadlock (8.0%) make up half of the verified properties. However, the other half consists of a variety of different properties, including consistency, fitness, determinism, interference freedom, responsiveness, mismatch, and loss-tolerance. This set of properties seems to be specific to the area of self-adaptation. Further research is required to obtain a solid understanding of the nature of these properties.

Fig. 9 shows the type of formal approach that are used in the study.

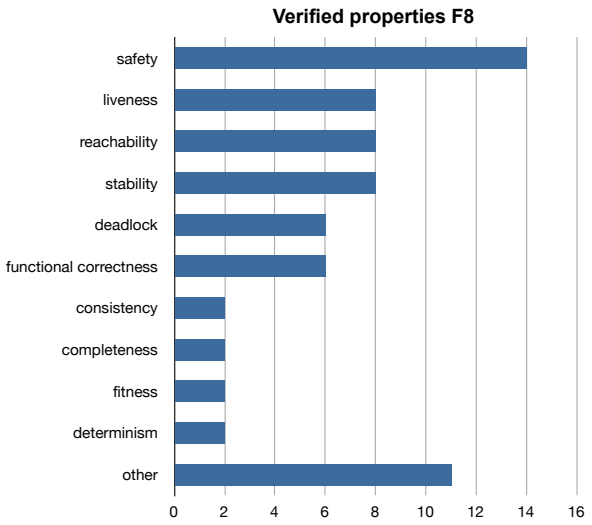

Figure 8: Verified properties of formalization

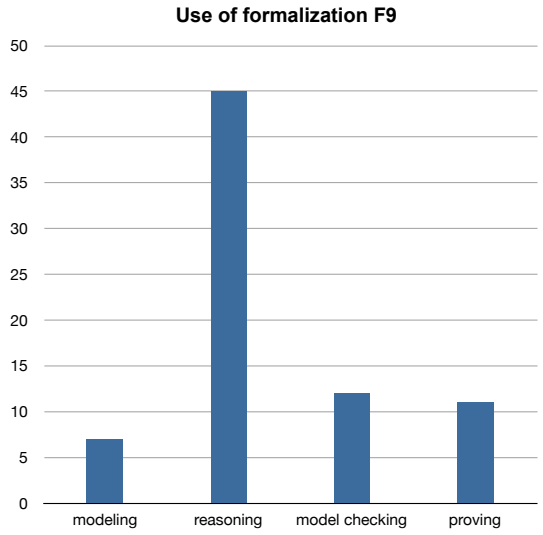

Figure 9: Formal approach used in the study

The majority of the studies use formal methods for reasoning purpose, i.e., 60.0%. In particular, most authors use a formal specification to reason about the design of a self-adaptive system. Model checking makes up 16.0% of the studies, and proving 14.7%. It is remarkable to see that in total, only 23 studies employ formal methods to actually provide evidence for the self-adaptive concerns of interest.

For data item F10, we found that 66.7% of the studies use formal methods for offline activities, while 33.3% use formal methods at runtime. Almost all studies use formal methods in one particular phase in the life cycle. One exception that employs formal results from design to check the implementation is [94].

As model checking is one of the prominent approaches to provide evidence for system properties, we analyzed the studies that employ model checking techniques, including [21, 36, 7, 94, 48, 63, 20, 41, 40]. We also included the recent publication [24]. From this analyzes, we derived an interesting model that maps the different types of behaviors of self-adaptive systems to zones of the state space. Fig. 10 shows an overview of the model.

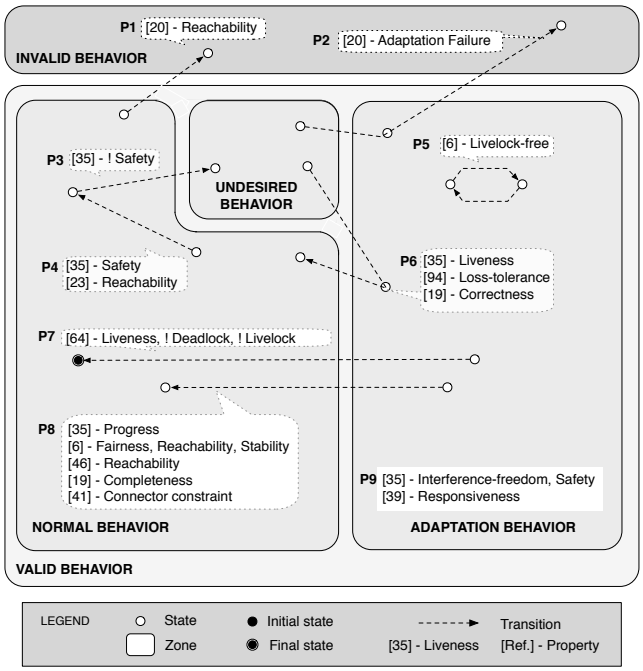

Figure 10: Zones in the state space that represent different behaviors of a self-adaptive system with properties of interest.

In the zone *normal behavior*, the system is performing its regular domain functionality. In the zone *undesired behavior*, the system requires adaptation for a concern of interest (e.g., a failure has occurred). In the zone *adaptive behavior*, the system is adapting itself to deal with the undesired behavior. Finally, the zone *invalid behavior* corresponds to states where the system should never be (e.g., the system is in a deadlock).

Properties of interest with respect to self-adaptation typically map to transitions between different zones. For example, property P1 in Fig. 10 checks whether invalid states can be reached from a system's normal behavior. One of the properties of P6 checks whether a system adapts correctly from undesired behavior via adaptation behavior back to normal behavior. One of the properties of P9 checks interference freedom, i.e., it verifies whether different adaptive behaviors do not interfere with one another.

5. LIMITATIONS OF THE STUDY

We have conducted a systematic literature review involving studies published in 16 venues from 2000 until 2011. As the set of venues is limited, there is a threat to validity with respect to the generalization of the conclusions of the study. We have anticipated this threat by taking into account all the primary publication venues for self-adaptive systems as well as the main conferences and journals on software engineering.

Another threat to validity is the existence of bias of the reviewers. To reduce this bias, we have taken several measures. First, we have established the protocol for the study during the planning phase. The protocol defines in advance how the systematic review is to be conducted, and this document was reviewed by an external reviewer. In addition, we have opted to let two reviewers perform the manual search independently. Then the results were compared and discussed in case of conflicts. To avoid bias in collecting data we

followed a similar approach. While this approach increases the validity of the results, it also significantly increased the review work.

6. FUTURE RESEARCH DIRECTIONS

To conclude, we outline ideas for future research derived from our study.

Formal methods in the field of self-adaptation are mainly used for modeling and reasoning (read discussing) about properties of interest. However, the full power of formal methods comes with automation, in particular for model checking and theorem proving. Tools are currently under-exploited and provide an opportunity to further mature the field.

We notice that a relevant part of the studies use formal methods at runtime. The main use of these methods is for modeling and analysis. There is plenty of room for research on exploiting the use of formal methods for other activities of self-adaptation. Futhermore, there is a need for pluggable light-weight tools that support efficient verification at runtime.

Particularly relevant for self-adaptive system is that work products of formalization are exploited throughout the software life cycle. This enables the transfer of quality assurances of self-adaptive systems obtained during design to the implementation and the running system, enhancing the validity of the required qualities [88]. Some studies that directly transfer formalization results over different phases of the software life cycle are [28, 82, 94, 87].

Researchers and engineers use standard languages for formal modeling and property specification. However, a set of new properties emerge that are subject of reasoning and verification, such as consistency, fitness, determinism, interference freedom, responsiveness, mismatch, and loss-tolerance. These properties appear to be specific to self-adaptation and are crucial to provide guarantees about concerns of self-adaptive systems. Developing a solid understanding and underpinning of these properties is an important subject for future research. The zone-based model presented in this paper offers an interesting starting point to get a better understanding of the specific nature of model-checking of self-adaptive systems.

We conclude with a final remark concerning the underpinning of formal papers in the area of self-adaptive systems. In many studies, authors introduce custom modeling language constructs. Often, the mathematical underpinning and soundness of these languages is assumed for granted (but usually not provided). This aspects requires attention as mathematical underpinning is the foundation of any formal method.

7. REFERENCES

[1] Y. Al-Nashif, A. Kumar, S. Hariri, Q. Guangzhi, L. Yi, and F. Szidarovsky. Multi-level intrusion detection system (ml-ids). In *International Conference on Automatic Computing*, 2008.

[2] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian. Resource management in the autonomic service-oriented architecture. In *International Conference on Autonomic Computing*, 2006.

[3] J. Gueyoung andK. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. Generating adaptation policies for multi-tier applications in consolidated server environments. In *International Conference on Automatic Computing*, 2008.

[4] C. Baier and J.P.Katoen. *Principles of Model Checking*. MIT Press, 2008.

[5] D. Balasubramaniam, R. Morrison, K. Mickan, G. Kirby, B. Warboys, I. Robertson, B. Snowdon, M. Greenwood, and W. Seet. Support for feedback and change in self-adaptive systems. In *Workshop on Self-Healing systems*, 2004.

[6] L. Baresi and L. Pasquale. Live goals for adaptive service compositions. In *Software Engineering for Adaptive and Self-Managing Systems*, 2010.

[7] B. Bartels and M. Kleine. A csp-based framework for the specification, verification, and implementation of adaptive systems. In *Software Engineering for Adaptive and Self-Managing Systems*, 2011.

[8] V. Basili, G. Caldiera, and D. Rombach. Goal question metric approach. In *Encyclopedia of Soft. Engineering*. 1994.

[9] B. Becker, D. Beyer, H. Giese, F. Klein, and D. Schilling. Symbolic invariant verification for systems with dynamic structural adaptation. In *28th International Conference on Software Engineering*, 2006.

[10] M. Ben, N. Georgantas, and V. Issarny. COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support. *Journal of Systems and Software*, 80(12):1941–1955, 2007.

[11] J. Bisbal and B. Cheng. Resource-based approach to feature interaction in adaptive software. In *Workshop on Self-Healing systems*, 2004.

[12] K. Biyani and S. Kulkarni. Mixed-mode adaptation in distributed systems: A case study. In *Software Engineering for Adaptive and Self-Managing Systems*, 2007.

[13] R. Borges, A. d'Avila Garcez, and L. Lamb. Integrating model verification and self-adaptation. In *International Conference on Automated Software Engineering*, 2010.

[14] A. Bracciali, A. Brogi, and C. Canal. A formal approach to component adaptation. *Journal of Systems and Software*, 74(1):45–54, 2005.

[15] J. Bradbury, J. Cordy, J. Dingel, and Wermelinger M. A survey of self-management in dynamic software architecture specifications. In *Workshop on Self-Managed Systems*, 2004.

[16] D. Breitgand, E. Henis, and O. Shehory. Automated and adaptive threshold setting: Enabling technology for autonomy and self-management. In *International Conference on Automatic Computing*, 2005.

[17] P. Brereton, B. Kitchenham, D. Budgen, M. Turner, and M. Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *Journal on Systems and Software*, 80, 2007.

[18] G. Brown, B. Cheng, H. Goldsby, and J. Zhang. Goal-oriented specification of adaptation requirements engineering in adaptive systems. In *Software Engineering for Adaptive and Self-Managing Systems*, 2006.

[19] O. Brukman, S. Dolev, and E. Kolodner. A self-stabilizing autonomic recoverer for eventual Byzantine software. *Journal of Systems and Software*, 81(12):2315–2327, 2008.

[20] A. Bucchiarone, C. Vattani P. Pelliccione, and O. Runge. Self-repairing systems modeling and verification using agg. Working IEEE/IFIP Conference on Software Architecture, 2009.

[21] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli. Dynamic qos management and optimization in service-based systems. *IEEE Transactions on Software Engineering*, 37(3):387–409, 2011.

[22] R. Calinescu and M. Kwiatkowska. Using quantitative analysis to implement autonomic it systems. In *31st International Conference on Software Engineering*, 2009.

[23] J. Camara, C. Canal, and G. Salaun. Behavioural self-adaptation of services in ubiquitous computing

environments. In *Software Engineering for Adaptive and Self-Managing Systems*, 2009.

[24] J. Camara and R. de Lemos. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In *Software Engineering for Adaptive and Self-Managing Systems*, 2012.

[25] C. Canal, P. Poizat, and G. Salaün. Model-based adaptation of behavioral mismatching components. *IEEE Transactions on Software Engineering*, 34(4):546–563, 2008.

[26] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, 2003.

[27] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. Lo Presti, and R. Mirandola. Moses: A framework for qos driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering*, 99, 2011.

[28] L. Cavallaro, E. Di Nitto, P. Pelliccione, M. Pradella, and M. Tivoli. Synthesizing adapters for conversational web-services from their wsdl interface. In *Software Engineering for Adaptive and Self-Managing Systems*, 2010.

[29] Luca Cavallaro and Elisabetta Di Nitto. An approach to adapt service requests to actual service interfaces. In *Software Engineering for Adaptive and Self-Managing Systems*, 2008.

[30] T. Chaari, D. Ejigu, F. Laforest, and V.M. Scuturici. A comprehensive approach to model and use context for adapting applications in pervasive environments. *Journal of Systems and Software*, 80(12):1973–1992, 2007.

[31] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*. LNCS vol. 5525, Springer, 2009.

[32] S.N. Chuang and A. Chan. Dynamic qos adaptation for mobile middleware. *IEEE Transactions on Software Engineering*, 34(6):738–752, 2008.

[33] F. Cuadrado, J. Duenas, and R. Garcia-Carmona. An autonomous engine for services configuration and deployment. *IEEE Transactions on Software Engineering*, 38:520–536, 2012.

[34] R. de Lemos, H. Giese, H. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. Goeschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzè, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J.P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*. LNCS, Springer, 2012.

[35] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. *Transactions on Autonomous and Adaptive Systems*, 1:223–259, 2006.

[36] A. Ebnenasir. Designing run-time fault-tolerance using dynamic updates. In *Software Engineering for Adaptive and Self-Managing Systems*, 2007.

[37] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *31st International Conference on Software Engineering*, 2009.

[38] N. Esfahani, E. Kouroshfar, and S. Malek. Taming uncertainty in self-adaptive software. In *International Symposium on Foundations of Software Engineering*, 2011.

[39] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *International Conference on Automated Software Engineering*, 2011.

[40] J. Fox. A formal orchestration model for dynamically adaptable services with cows. In *International Conference on Adaptive and Self-Adaptive Systems and Applications*, 2011.

[41] D. Garlan and B. Schmerl. Model-based adaptation for self-healing systems. In *Workshop on Self-Healing systems*, 2002.

[42] David Garlan, S.W. Cheng, A. Cheng Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 37:46–54, 2004.

[43] S. Gilbert, N. Lynch, S. Mitra, and T. Nolte. Self-stabilizing robot formations over unreliable networks. *ACM Transactions on Autonomous and Adaptive Systems*, 4(3):17:1–17:29, 2009.

[44] M. Güdemann, F. Nafz, F. Ortmeier, H. Seebach, and W. Reif. A specification and construction paradigm for organic computing systems. In *International Conference on Self-Adaptive and Self-Organizing Systems*, 2008.

[45] J. Guofei, C. Haifeng, and K. Yoshihira. Discovering likely invariants of distributed transaction systems for autonomic system management. In *International Conference on Autonomic Computing*, 2006.

[46] R. Haesevoets, D. Weyns, T. Holvoet, and W. Joosen. A formal model for self-adaptive and self-healing organizations. In *Software Engineering for Adaptive and Self-Managing Systems*, 2009.

[47] K. Hansen, W. Zhang, and M. Ingstrup. Towards self-managed executable petri nets. In *International Conference on Self-Adaptive and Self-Organizing Systems*, 2008.

[48] W. Heaven, D. Sykes, J. Magee, and J. Kramer. Software engineering for adaptive and self-managing systems. chapter A case study in goal-driven architectural adaptation. 2009.

[49] D. Herbert, V. Sundaram, Y.H. Lu, S. Bagchi, and Z. Li. Adaptive correctness monitoring for wireless sensor networks using hierarchical distributed run-time invariant checking. *ACM Transactions on Autonomous and Adaptive Systems*, 2(3), 2007.

[50] M. Huebscher and J. McCann. A survey of autonomic computing: degrees, models, and applications. *ACM Computer Surveys*, 40:7:1–7:28, 2008.

[51] A. Joolia, T. Batista, G. Coulson, and A. Gomes. Mapping adl specifications to an efficient and reconfigurable runtime component platform. In *5th Working IEEE/IFIP Conference on Software Architecture*, 2005.

[52] M. Karlsson and M. Covell. Dynamic black-box performance model estimation for self-tuning regulators. In *International Conference on Automatic Computing*, 2005.

[53] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[54] M. Kim, J. Jeong, and S. Park. From product lines to self-managed systems: an architecture-based runtime reconfiguration framework. In *Design and Evolution of Autonomic Application Software*, 2005.

[55] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. (EBSE 2007-001, Keele and Durham University), 2007.

[56] S. Ko, I. Gupta, and Y. Jo. A new class of nature-inspired algorithms for self-adaptive peer-to-peer computing. *ACM Transactions on Autonomous and Adaptive Systems*, 3(3):11:1–11:34, 2008.

[57] L. Konig and H. Schmeck. A completely evolvable genotype-phenotype mapping for evolutionary robotics. In *International Conference on Self-Adaptive and Self-Organizing Systems*, 2009.

[58] J. Kramer and J. Magee. Self-managed systems: An architectural challenge. *Future of Software Engineering*, 2007.

[59] A. Lapouchnian, W. Robinson, V. Souza, and J. Mylopoulos. Awareness requirements for adaptive systems. Software Engineering for Adaptive and Self-Managing Systems, 2011.

[60] Z. Lei and N. Georganas. Adaptive video transcoding and streaming over wireless channels. *Journal of Systems and Software*, 75(3):253–270, 2005.

[61] J. Magee and T. Maibaum. Towards specification, modelling and analysis of fault tolerance in self managed systems. In *Software Engineering for Adaptive and Self-Managing Systems*, 2006.

[62] S.S. Manvi and P. Venkataram. An agent based adaptive bandwidth allocation scheme for multimedia applications. *Journal of Systems and Software*, 75(3):305–318, 2005.

[63] R. Mateescu, P. Poizat, and G. Salaün. Behavioral adaptation of component compositions based on process algebra encodings. In *International Conference on Automated Software Engineering*, 2007.

[64] R. Mateescu, P. Poizat, and G. Salaun. Adaptation of service protocols using process algebra and on-the-fly reduction techniques. *IEEE Transactions on Software Engineering*, 99, 2011.

[65] B. Morin, O. Barais, G. Nain, and J.M. Jezequel. Taming dynamically adaptive systems using models and aspects. In *31st International Conference on Software Engineering*, 2009.

[66] R. Nou, S. Kounev, F. Julià, and J. Torres. Autonomic QoS control in enterprise Grid environments using online simulation. *Journal of Systems and Software*, 82(3):486–502, 2009.

[67] P. Oreizy, N. Medvidovic, and R. Taylor. Architecture-based runtime software evolution. In *20th International Conference on Software engineering*. IEEE, 1998.

[68] P. Pelliccione, M. Tivoli, A. Bucchiarone, and A. Polini. An architectural approach to the correct and automatic assembly of evolving component-based systems. *Journal of Systems and Software*, 81(12):2237–2251, 2008.

[69] H. Raja and O. Scholz. A case study on self-sufficiency of individual robotic modules in an arena with limited energy resources. In *International Conference on Adaptive and Self-Adaptive Systems and Applications*, 2011.

[70] C. Roblee and G. Cybenko. Implementing large-scale autonomic server monitoring using process query systems. In *International Conference on Automatic Computing*, 2005.

[71] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *Transactions on Autonomous and Adaptive Systems*, 4:14:1–14:42, 2009.

[72] M. Sama, S. Elbaum, F. Raimondi, D. Rosenblum, and Z. Wang. Context-aware adaptive applications: Fault patterns and their automated identification. *IEEE Transactions on Software Engineering*, 36(5):644–661, 2010.

[73] M. Sama, D. Rosenblum, Z. Wang, and S. Elbaum. Model-based fault detection in context-aware adaptive applications. In *International Symposium on Foundations of Software Engineering*, 2008.

[74] T. Seceleanu and D. Garlan. Developing adaptive systems with synchronized architectures. *Journal of Systems and Software*, 79(11):1514–1526, 2006.

[75] S. Seshia. Autonomic reactive systems via online learning. In *International Conference on Autonomic Computing*, 2007.

[76] H. Shan, G. Jiang, and K. Yoshihira. Extracting overlay invariants of distributed systems for autonomic system management. In *International Conference on Self-Adaptive and Self-Organizing Systems*, 2010.

[77] B. Solomon, D. Ionescu, M. Litoiu, and G. Iszlai. Autonomic computing control of composed web services. In *Software Engineering for Adaptive and Self-Managing Systems*, 2010.

[78] Y. Steven, G. Indranil, and J. Yookyung. Novel mathematics-inspired algorithms for self-adaptive peer-to-peer computing. In *International Conference on Self-Adaptive and Self-Organizing Systems*, 2007.

[79] D. Sykes, W. Heaven, J. Magee, and J. Kramer. From goals to components: a combined approach to self-management. In *Software Engineering for Adaptive and Self-Managing Systems*, 2008.

[80] D. Tacconi, D. Miorandi, I. Carreras, F. D. Pellegrini, and I. Chlamtac. Cooperative evolution of services in ubiquitous computing environments. *ACM Transactions on Autonomous and Adaptive Systems*, 6(3):20:1–20:24, 2011.

[81] G. Tamura, N.M. Villegas, H. Muller, J.P. Sousa, B. Becker, G. Karsai, S. Mankovskii, M. Pezze, W. Schafer, L. Tahvildari, and K. Wong. Towards practical runtime verification and validation of self-adaptive software systems. *Software Engineering for Self-Adaptive Systems II, Springer*, 2012.

[82] L. Tan. Model-based self-monitoring embedded programs with temporal logic specifications. In *International Conference on Automated Software Engineering*, 2005.

[83] E. Vassev and M. Hinchey. ASSL: A software engineering approach to autonomic computing. *Computer*, 42(6):90–93, 2009.

[84] N. Villegas, H. Müller, G. Tamura, L. Duchien, and R. Casallas. A framework for evaluating quality-driven self-adaptive software systems. In *Software Engineering for Adaptive and Self-Managing Systems*, 2011.

[85] X. Wang, Z. Du, Y. Chen, and S. Li. Virtualization-based autonomic resource management for multi-tier Web applications in shared data center. *Journal of Systems and Software*, 81(9):1591–1608, 2008.

[86] I. Warren, J. Sun, S. Krishnamohan, and T. Weerasinghe. An automated formal approach to managing dynamic reconfiguration. 2006.

[87] M. Wermelinger, A. Lopes, and J. Fiadeiro. A graph based architectural (re)configuration language. In *International Symposium on Foundations of Software Engineering*, 2001.

[88] D. Weyns. Towards an integrated approach for validating

qualities of self-adaptive systems. *ISSTA Workshop on Dynamic Analysis*, 2012.

[89] D. Weyns, U. Iftikhar, S. Malek, and J. Andersson. Claims and supporting evidence for self-adaptive systems. *Software Engineering for Adaptive and Self-Managing Systems*, 2012.

[90] D. Weyns, S. Malek, and J. Andersson. Forms: a formal reference model for self-adaptation. In *International Conference on Automatic Computing*, 2010.

[91] D. Weyns, S. Malek, and J. Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1), 2012.

[92] C. Ye, S. Cheung, and W. Chan. Process evolution with atomicity consistency. In *Software Engineering for Adaptive and Self-Managing Systems*, 2007.

[93] Z. Yu, J. Tsai, and T. Weigert. An adaptive automatically tuning intrusion detection system. *ACM Transactions on Autonomous and Adaptive Systems*, 3(3):10:1–10:25, 2008.

[94] J. Zhang and B. Cheng. Model-based development of dynamically adaptive software. In *28th International Conference on Software Engineering*, 2006.

[95] J. Zhang and B. Cheng. Using temporal logic to specify adaptive program semantics. *Journal of Systems and Software*, 79(10):1361–1369, 2006.

[96] Z. Zhang and H. Shen. M-aid: An adaptive middleware built upon anomaly detectors for intrusion detection and rational response. *ACM Transactions on Autonomous and Adaptive Systems*, 4(4):24:1–24:35, 2009.

Appendix A

ID	Year	Title	Author
SEAMS [79]	2008	From Goals To Components: A Combined Approach To Self-Management	Sykes et al.
SEAMS [23]	2009	Behavioural Self-Adaptation of Services in Ubiquitous Computing Environments	Camara et al.
SEAMS [46]	2009	A Formal Model for Self-Adaptive and Self-Healing Organizations	Haesevoets et al.
SEAMS [6]	2010	Live Goals for Adaptive Service Compositions	Baresi & Pasquale
SEAMS [28]	2010	Synthesizing adapters for conversational web-services from their WSDL interface	Cavallaro et al.
SEAMS [77]	2010	Autonomic Computing Control of Composed Web Services	Solomon et al.
SEAMS [59]	2011	Awareness Requirements for Adaptive Systems	Souza
SEAMS [7]	2011	A CSP-based Framework for the Specification, Verification, and Implementation of Adaptive Systems	Bartels & Kleine
SEAMS [48]	2009	A Case Study in Goal-Driven Architectural Adaptation	Heaven et al.
SEAMS [18]	2006	Goal-oriented Specification of Adaptation Requirements Engineering in Adaptive Systems	Brown et al.
SEAMS [61]	2006	Towards Specification, Modelling and Analysis of Fault Tolerance in Self Managed Systems	Magee & Maibaum
SEAMS [12]	2007	Mixed-Mode Adaptation in Distributed Systems: A Case Study	Biyani & Kulkarni
SEAMS [36]	2007	Designing Run-Time Fault-Tolerance Using Dynamic Updates	Ebnenasir
SEAMS [92]	2007	Process Evolution with Atomicity Consistency	Ye & Chan
SEAMS [29]	2008	An Approach to Adapt Service Requests to Actual Service Interfaces	Cavallaro & Nitto
SASO [76]	2010	Extracting Overlay Invariants of Distributed Systems for Autonomic System Management	Shan et al.
SASO [47]	2008	Towards Self-Managed Executable Petri Nets	Hansen et al.
SASO [44]	2008	A Specification and Construction Paradigm for Organic Computing Systems	Güdemann et al.
SASO [78]	2007	Novel Mathematics-Inspired Algorithms for Self-Adaptive Peer-to-Peer Computing	Ko et al.
SASO [57]	2009	A Completely Evolvable Genotype-Phenotype Mapping for Evolutionary Robotics	König & Schmeck
WICSA [51]	2005	Mapping ADL Specifications to an Efficient and Reconfigurable Runtime Component Platform	Joolia et al.
WICSA [20]	2009	Self-Repairing Systems Modeling & Verification using AGG	Bucchiarone et al.
WOSS [5]	2004	Support for Feedback and Change in Self-adaptive Systems	Balasubramaniam et al.
WOSS [41]	2002	Model-based Adaptation for Self-Healing Systems	Garlan & Schmerl
WOSS [11]	2004	Resource-based Approach to Feature Interaction in Adaptive Software	Bisbal & Cheng
FSE [73]	2008	Model-Based Fault Detection in Context-Aware Adaptive Applications	Sama et al.
FSE [38]	2011	Taming Uncertainty in Self-Adaptive Software	Esfahani et al.
FSE [87]	2001	A Graph Based Architectural (Re)configuration Language	Wermelinger et al.
JSS [66]	2009	Autonomic QoS control in enterprise Grid environments using online simulation	Nou et al.
JSS [74]	2006	Developing adaptive systems with synchronized architectures	Seceleanu & Garlan
JSS [14]	2005	A formal approach to component adaptation	Bracciali et al.
JSS [95]	2006	Using temporal logic to specify adaptive program semantics	Zhang & Cheng
JSS [68]	2008	An architectural approach to the correct and automatic assembly of evolving component-based systems	Pelliccione et al.
JSS [19]	2008	A self-stabilizing autonomic recoverer for eventual Byzantine software	Brukman et al.
JSS [85]	2008	Virtualization-based autonomic resource management for multi-tier Web applications in shared data center	Wang et al.
JSS [10]	2007	COCOA: COnversation-based service COmposition in pervAsive computing environments with QoS support	Mokhtar et al.
JSS [62]	2005	An agent based adaptive bandwidth allocation scheme for multimedia applications	Manvi & Venkataram
JSS [60]	2005	Adaptive video transcoding and streaming over wireless channels	Z. Lei & Georganas
JSS [30]	2007	A comprehensive approach to model and use context for adapting applications in pervasive environments	Chaari et al.

ID	Year	Title	Author
DEAS [54]	2005	From Product Lines to Self-Managed Systems: An Architecture-Based Runtime Reconfiguration Framework	Kim et al.
ICSE [37]	2009	Model Evolution by Run-Time Parameter Adaptation	Epifani
ICSE [22]	2009	Using Quantitative Analysis to Implement Autonomic IT Systems	Calinescu & Kwiatkowska
ICSE [65]	2009	Taming Dynamically Adaptive Systems Using Models and Aspects	Morin et al.
ICSE [9]	2006	Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation	Becker et al.
ICSE [94]	2006	Model-Based Development of Dynamically Adaptive Software	Zhang & Cheng
TSE [26]	2003	CARISMA: Context-Aware Reflective mIddleware System for Mobile Applications	Capra et al.
TSE [64]	2011	Adaptation of Service Protocols using Process Algebra and On-the-Fly Reduction Techniques	Mateescu et al.
TSE [32]	2008	Dynamic QoS Adaptation for Mobile Middleware	Chuang et al.
TSE [21]	2011	Dynamic QoS Management and Optimization in Service-Based Systems	Calinescu et al.
TSE [33]	2011	An Autonomous Engine for Services Configuration and Deployment	Cuadrado et al.
TSE [25]	2008	Model-Based Adaptation of Behavioral Mismatching Components	Canal et al.
TSE [72]	2010	Context-Aware Adaptive Applications: Fault Patterns and Their Automated Identification	Sama et al.
TSE [27]	2011	MOSES: a Framework for QoS Driven Runtime Adaptation of Service-oriented Systems	Cardellini et al.
ASE [63]	2007	Behavioral Adaptation of Component Compositions based on Process Algebra Encodings	Mateescu et al.
ASE [39]	2011	Self-Adaptive Software Meets Control Theory: A Preliminary Approach Supporting Reliability Requirements	Filieri et al.
ASE [13]	2010	Integrating Model Verification and Self-Adaptation	Borges et al.
ASE [82]	2005	Model-Based Self-Monitoring Embedded Programs With Temporal Logic Specifications	Tan
ASE [86]	2006	An Automated Formal Approach to Managing Dynamic Reconfiguration	I. Warren et al.
Adaptive [69]	2011	A Case Study on Self-Sufficiency of Individual Robotic Modules in an Arena With Limited Energy Resources	R. Humza & Scholz
Adaptive [40]	2011	A Formal Orchestration Model for Dynamically Adaptable Services with COWS	Fox
TAAS [56]	2008	A New Class of Nature-Inspired Algorithms for Self-Adaptive Peer-to-Peer Computing	Ko & Gupta
TAAS [96]	2009	M-AID: An Adaptive Middleware Built Upon Anomaly Detectors for Intrusion Detection and Rational Response	Zhang
TAAS [43]	2009	Self-Stabilizing Robot Formations over Unreliable Networks	Gilbert et al.
TAAS [93]	2008	An Adaptive Automatically Tuning Intrusion Detection System	Yu & Tsai
TAAS [49]	2007	Adaptive Correctness Monitoring for Wireless Sensor Networks Using Hierarchical Distributed Run-Time Invariant Checking	Herbert et al.
TAAS [80]	2011	Cooperative Evolution of Services in Ubiquitous Computing Environments	Tacconi
ICAC [45]	2006	Discovering Likely Invariants of Distributed Transaction Systems for Autonomic System Management	Jiang
ICAC [2]	2006	Resource Management in the Autonomic Service-Oriented Architecture	Almeida et al.
ICAC [70]	2005	Implementing Large-Scale Autonomic Server Monitoring Using Process Query Systems	Roblee et al.
ICAC [52]	2005	Dynamic Black-Box Performance Model Estimation for Self-Tuning Regulators	Karlsson & Covell
ICAC [75]	2007	Autonomic Reactive Systems via Online Learning	Seshia
ICAC [90]	2010	FORMS: a FOrmal Reference Model for Self-adaptation	Weyns et al.
ICAC [16]	2005	Automated and Adaptive Threshold Setting: Enabling Technology for Autonomy and Self-Management	D. Breitgand et al.
ICAC [1]	2008	Multi-Level Intrusion Detection System (ML-IDS)	Al-Nashif et al.
ICAC [3]	2008	Generating Adaptation Policies for Multi-Tier Applications in Consolidated Server Environments	Jung et al.