A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems

Norha M. Villegas
Hausi A. Müller
Dept. of Computer Science
University of Victoria
Victoria, Canada
{nvillega,hausi}@cs.uvic.ca

Gabriel Tamura
Laurence Duchien
INRIA Lille-Nord Europe
University of Lille 1
Lille, France
First.Lastname@inria.fr

Rubby Casallas
Dept. of Computer Science
University of Los Andes
Bogotá, Colombia
rcasalla@uniandes.edu.co

ABSTRACT

Over the past decade the dynamic capabilities of self-adaptive software-intensive systems have proliferated and improved significantly. To advance the field of self-adaptive and self-managing systems further and to leverage the benefits of self-adaptation, we need to develop methods and tools to assess and possibly certify adaptation properties of self-adaptive systems, not only at design time but also, and especially, at run-time. In this paper we propose a framework for evaluating quality-driven self-adaptive software systems. Our framework is based on a survey of self-adaptive system papers and a set of adaptation properties derived from control theory properties. We also establish a mapping between these properties and software quality attributes. Thus, corresponding software quality metrics can then be used to assess adaptation properties.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*reliability, validation*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Measurement, performance, reliability, security, verification

Keywords

Software adaptation properties, software adaptation metrics, assessment and evaluation of self-adaptive systems, software quality attributes, application of control theory, engineering of self-adaptive systems, run-time validation and verification

1. INTRODUCTION

Over the past decade, self-adaptation has increasingly become a fundamental concern in the engineering of software systems to reduce the high costs of software maintenance and evolution and to regulate the satisfaction of functional and extra-functional requirements under changing conditions of system execution. Even though adaptation mechanisms have been widely investigated in the engineering of dynamic software systems, their application to real problems is still limited due to a lack of methods for validation and verification of complex, adaptive, nonlinear applications [26]. After an extensive analysis of self-adaptive approaches, we concluded that adaptation properties and the corresponding metrics are rarely identified or explicitly addressed in papers dealing with the engineering of dynamic software systems. Consequently, without explicit adaptation properties it is impossible to assess and certify adaptive system behavior. In light of this, evaluation techniques, such as run-time validation and verification, are needed to advance the field.

To leverage the capabilities of self-adaptive systems, it is necessary to validate adaptation mechanisms to ensure that self-adaptive software systems function properly and users can trust them. To address this problem we propose a framework for evaluating self-adaptive systems, where adaptation properties are specified explicitly and driven by quality attributes, such as those defined by researchers at Software Engineering Institute (SEI) [2]. Our framework provides (i) a set of dimensions useful to classify self-adaptive systems; (ii) a compendium of adaptation properties to be observed in control loop implementations (in terms of the controller and the managed system); (iii) a mapping of adaptation properties to quality attributes to evaluate adaptation properties; and (iv) a set of quality metrics to evaluate adaptation properties and quality attributes. To define adaptation properties we analyzed existing self-adaptive approaches and investigated properties used in control theory. We then established a mapping between adaptation properties and software quality attributes. Finally, we identified a set of metrics used to evaluate software quality attributes. The mapping of adaptation properties to quality attributes and their corresponding quality metrics constitute our framework.

Borrowing properties and metrics from control theory and re-interpreting them for self-adaptive software is not a trivial task. On one side, the semantics of the concepts involved in adaptation for control theory are different than those for self-adaptive software. On the other side, existing self-adaptive software approaches do not generally address adaptation properties explicitly. Yet another important challenge is that, in general, self-adaptive software systems are nonlinear systems [12].

SEAMS'11, May 23–24, 2011, Waikiki, Honolulu, HI, USA

Metrics to evaluate feedback control systems depend on the properties that result from the relationship between the control objectives, the target system's measured outputs, the disturbances affecting the system and how the target system is considered in the adaptation strategy [11]. Our reinterpretation of these properties and metrics results from the analysis of several representative approaches and strategies that have been proposed to achieve behavior modification in a managed system. From the identified relationship and the analysis of these strategies we identified two main dimensions to classify and evaluate self-adaptive software. These dimensions arise from the way the strategy, *structural* or *behavioral*, addresses (i) the managed system (i.e., the system to be controlled), and (ii) the controller itself.

Some previous papers have addressed the evaluation of self-adaptive software. Meng proposed a mapping of fundamental concepts from control theory to self-adaptive software systems [17]. In his vision, the fundamental properties to evaluate self-adaptation are stability and robustness. These two properties are analyzed and characterized in terms of what they imply for the programming paradigms, architectural styles, modeling paradigms, and software engineering principles. However, his evaluation model is descriptive and not being applied to any approach. In the taxonomy proposed by Salehie and Tahvildari, several representative projects addressing the adaptation of software systems were surveyed in terms of a set of adaptation concerns: *how, what, when* and *where* [22]. They also proposed a hierarchical view of self-* properties and discussed their relationship with quality factors of software systems. However, the scope of their work did not include the identification of metrics for evaluating self-adaptive software systems in light of the identified adaptation properties and quality attributes. Our contribution in this paper differs from the aforementioned in the following aspects. First, we provide a definition of a more comprehensive list of properties found in control theory and apply this list to the analysis of self-adaptive software systems. Second, none of the studied contributions presents a comprehensive and unified list of adaptation properties applicable to software systems. Third, we provide a valuable foundation to evaluate adaptation properties in terms of quality attributes and metrics as widely practiced in the engineering of software systems.

The remainder of this paper is organized as follows. Section 2 presents our proposed model to characterize and classify self-adaptive software. Section 3 presents the analysis of selected adaptive systems based on the characterization model presented in Sect. 2. Section 4 presents a compendium of the metrics and properties found in the analyzed approaches with their corresponding definitions. We also illustrate our proposed mapping between adaptive properties and quality attributes and how this mapping can be used to evaluate adaptive systems. Section 5 discusses findings of our analysis and open challenges. Finally, Section 6 concludes the paper.

2. A CHARACTERIZING MODEL FOR SELF-ADAPTIVE SOFTWARE

In this section we propose a model consisting of eight analysis dimensions to characterize self-adaptive software. This model constitutes a foundation for evaluating self-adaptive systems. For each of the analysis dimensions, the model considers a set of standardized classification options. These options resulted from combining classification attributes from recognized authoritative sources (e.g., SEI) with those found in the set of papers that we analyzed. In particular, the set of options for the analyzed quality attributes as observable adaptation properties was identified mainly using the taxonomy proposed by an SEI study [2]. This taxonomy provides a comprehensive characterization of software quality attributes, their concerns, factors that affect them and methods for their evaluation. For each analysis dimension, we include the relevant options available from control theory, as follows.

Adaptation goal. This is the main reason or justification for the system or approach to be self-adaptive. Adaptation goals are usually defined through one or more of the self-* properties, the preservation of specific quality of service (QoS) properties, or the regulation of non-functional requirements in general.

Reference inputs. The concrete and specific set of values, and corresponding types, that are used to specify the state to be achieved and maintained in the managed system by the adaptation mechanism, under changing conditions of system execution. Reference inputs are specified as (a) single reference values (e.g., a physically or logically-measurable property); (b) some form of contract (e.g., quality of service (QoS), service level agreements (SLA), or service level objectives (SLO); (c) goal-policy-actions; (d) constraints defining computational states (according to the particular proposed definition of *state*); or even (e) functional requirements (e.g., logical expressions as invariants or assertions, regular expressions).

Measured outputs. The set of values, and corresponding types, that are measured in the managed system. Naturally, as these measurements must be compared to the reference inputs to evaluate whether the desired state has been achieved, it should be possible to find relationships between these inputs and outputs. Furthermore, we consider two aspects on the measured outputs: how they are specified and how monitored. For the specification, the identified options are (a) continuous domains for single variables or signals; (b) logical expressions or conditions for contract states; and (c) conditions expressing states of system malfunction. For monitoring, the options are (a) measurements on physical properties from physical devices (e.g., CPU temperature); (b) measurements on logical properties of computational elements (e.g., request processing time in software or CPU load in hardware); and (c) measurements on external context conditions (e.g., user localization or weather conditions).

Computed control actions. These are characterized in the monitor, analyze, plan, execute, and knowledge (MAPE-K) loop context, and in particular by the nature of the output of the adaptation planner or controller [13]. This output affects the managed system to have the desired effect. The computed control actions can be (a) continuous signals that affect behavioral properties of the managed system; (b) discrete operations affecting the computing infrastructure executing the managed system (e.g., host system's buffer allocation and resizing operations; modification of process scheduling in the CPU); (c) discrete operations that affect the processes of the managed system directly (e.g., processes-level service invocation, process execution operations—halt/resume, sleep/re-spawn/priority modification of processes); and (d) discrete operations affecting the

managed system's software architecture (e.g., managed system's architecture reconfiguration operations). The nature of these outputs is related to the extent of the intrusiveness of the adaptation mechanism with respect to the managed system and defines the extent of the adaptation mechanism to exploit the knowledge about either the structure or the behavior of the managed system in the adaptation process.

System structure. Self-adaptive systems have two well-defined subsystems (although possibly indistinguishable): (i) the adaptation controller and (ii) the managed system. One reason for analyzing controller and managed system structures is to identify whether a given approach implements the adaptation controller embedded within the managed system. Another reason is to identify the effect that the separation of concerns in these two subsystems has in the achievement of the adaptation goal. The analyzed approaches can be grouped into two sets: (i) those modeling the structure of the managed system to influence its behavior by modifying its structure; and (ii) those modeling the managed system's behavior to influence it directly. We consider the behavior model and the structure model as part of the managed system's structure. The identified options for the controller structure are variations of the MAPE-K loop with either behavioral or structural models of the managed system: (a) feedback control, that is, a MAPE-K structure with a fixed adaptation controller (e.g., a fixed set of transfer functions as a behavior model of the managed system) [11]; (b) adaptive control: a MAPE structure extended with managed system's reference or identification models of behavior (e.g., tunable parameters of controller for adaptive controllers: model reference adaptive control (MRAC) or model identification adaptive control (MIAC)) [19, 7]; (c) reconfigurable control: MAPE-K structure with modifiable controller algorithm (e.g., rule-based software architecture reconfiguration controller). For the managed system structure, the identified options are: (a) non-modifiable structure (e.g., monolithic system); and (b) modifiable structure with/without reflection capabilities (e.g., reconfigurable software components architecture). It is worth noting that not all options for system structure can be combined with any options for computed control actions. For instance, discrete operations affecting the computing infrastructure executing the managed system could be used to improve the performance of a monolithic system, whereas discrete operations for affecting this managed system's software architecture would not make sense.

Observable adaptation properties. By adaptation property we mean a quality (or characteristic) that is particular to a specific adaptation approach or mechanism. A quality can be a specific attribute value in a given state or a characteristic response to a known stimulus in a given context. Thus, observable adaptation properties are properties that can be identified and measured in the adaptation process. Given that we distinguish between the controller and the managed system in any self-adaptive system, we analyze observable adaptation properties also in both the controller and the managed system. The identified properties for the controller are (a) stability; (b) accuracy; (c) settling-time; (d) small-overshoot; (e) robustness; (f) termination; (g) consistency (in the overall system structure and behavior); (h) scalability; and (i) security. For the managed system, the identified properties result from the adaptation process: (a) behavioral/functional invariants; and (b) quality of service conditions, such as performance (i.e., latency, throughput, capacity); dependability (i.e., availability, reliability, maintainability, safety, confidentiality, integrity); security (i.e., confidentiality, integrity, availability); and safety (i.e, interaction complexity and coupling strength). Our proposed definitions for these properties are given in Sect. 4.2.

Proposed evaluation. For the analyzed approaches, we used this element to identify the strategies proposed to evaluate themselves. Among the most used evaluation mechanisms are the execution of tests in real or simulated execution platforms, and the illustration with example scenarios.

Identified metrics. Correspond to metrics that were used to measure the adaptation's variables of interest in the analyzed approaches.

3. ANALYSIS OF SELF-ADAPTIVE APPROACHES

To validate our model for evaluating self-adaptive systems, we analyzed over 20 published approaches dealing with such systems. Of course, developing the model and analyzing the subject systems was an iterative process. Table 2 presents the characterization of selected adaptive approaches based on the evaluation model proposed in Sect. 2. Note that two of the dimensions (i.e., columns *Adaptation Properties* and *Evaluation/Metrics*) are detailed in Sect. 4.

Self-adaptive approaches range from pure control theory approaches to pure software engineering-based approaches with many hybrid approaches in-between. In control theory-based approaches, control actions are continuous signals that affect behavioral parameters of the managed system. The structure of the managed system in these approaches is generally non-modifiable while its behavior is modeled mathematically [20]. In contrast, software engineering-based approaches are characterized by implementing discrete control actions that affect the managed system's software architecture (i.e., the system structure). In these approaches the adaptation is supported by a model of the managed system's structure and reflection capabilities that allow the modification of the system structure [1, 6, 9, 10, 14, 15, 18, 23, 25]. In hybrid adaptive systems, control actions are generally discrete operations that affect either the computing infrastructure executing the managed system or the set of processes comprising the managed system. Usually, the structure of the managed system is non-modifiable [3, 4, 5, 8, 27]. It might be useful to classify hybrid approaches in further. For instance, we classified the approach proposed by Solomon et al. [24] between hybrid and software engineering-based approaches, given that their control actions affect the architecture of the managed system but the analysis is based on a behavioral model of the managed system to decide when to adapt.

According to our proposed spectrum, most approaches were identified as software engineering-based and hybrid adaptive systems. With respect to the adaptation goal, we did not identify a relationship with our proposed spectrum. This means that it is possible to find any of the adaptation goals along the entire spectrum. Concerning reference inputs, most approaches use contracts as the way to specify reference values for the adaptation goal and the corresponding measured outputs. All approaches that explicitly addressed monitoring, do so by monitoring logical properties of computational elements (internal context), while two of them

Table 1: The characterization model applied to filtered adaptive approaches

Approach	Adaptation Goal	Reference Inputs	Measured Outputs	Control Actions	System Structure	Adaptation Properties	Evaluation/Metrics
Appleby et al. Océano [1]	Self-manag. Self-optim.	Contracts: SLAs	SLOs/Logical properties of computational elements	Discrete operations affecting the comput. infrastructure	Adaptive ctrl./Modif. struct.reflect.	Stab., settl. time, small overshoot, scalab./Scalab., Dependability: availab.maintainab.	Settl. time: perform. adapt.process./Active servers, connections, resp. time, output bwidth, throttle and admission rates
Baresi and Guinea [3]	Self-recov.	Contracts: SLAs, funct. req.	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's process	Adaptive ctrl./Non-modif.struct.	None for the controller/Behavioral, Dependab: safety, integrity, availab., reliab.	Functional and reliab./ Reliability>=0.95 $\frac{ResponseFrequency}{TimeUnit}$
Candea et al. Microreboots [4]	Self-recov.	Contracts: SLOs-QoS	Malfunct. condit./Logic. properties of comput. elem.	Discrete oper. managed system's process	Adaptive ctrl./Modif. struct.reflect.	Overshoot, settl. time/Dependab: availability	Recovery/Availability $A = \frac{MTTF}{MTTF+MTTR}$ Downtime $U = \frac{MTTR}{MTTF+MTTR}$
Cardellini et al. MOSES [5]	QoS preser.	Contracts: QoS	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's process	Reconfig. ctrl./Modif. struct.reflect.	Accuracy/ Perform.: latency, Depend.:reliab., cost.	Accuracy adap.strat./ Response time (Ru), execution cost (Cu), expected reliab. (Du)
Dowling and Cahill. K-Comp. [6]	Self-manag.	Contracts: SLOs-QoS	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's soft. architecture	Reconfig. ctrl./Modif. struct.reflect.	Robust., scalab./ Performance: througp., capac.	None/ Load cost of components
Ehrig et al. [8]	Self-healing	Goal actions	Malfunct. condit./Logic. properties of comput. elem.	Discrete oper. managed system's process	Adaptive ctrl./Non-modifiable structure	Termination/ Dependab:reliab.	Formal verification of properties. Running example/ None
Floch et al. MADAM [9]	QoS preser. Self-config.	Contracts: SLAs-SLOs-QoS	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's soft. archiecture	Adaptive ctrl./Modif. structure reflection	Scalab/Dependab: reliab., maintain.	Scalability Simulation environment/ None
Garlan et al. Rainbow [10]	Self-repair.	Contracts: SLAs-SLOs-QoS	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's soft. arch.	Adaptive ctrl./Modif. struct.reflect.	None for the controller/Perform: latency	Effectiveness. Running examp./ Latency (not explicitly defined)
Kumar et al. MWare [14]	Self-manag. Self-config. Self-optim.	Contracts: QoS; policy actions	SLOs/Logic. properties of comput. elem. ext. context	Discrete oper. managed system's soft. architecture	Adaptive ctrl./Modif. structure reflection	Settl. time, overshoot/ Performance: through., capac.	Real scenarios/ Business utility function
Léger et al. [15]	QoS preser. Self-config.	Constraints: comput. states	Malfunct. condit./Logic. properties of comput. elem.	Discrete oper. managed system's soft. architecture	Reconfig. ctrl./Modif. struct.reflect.	Consist.: atom., isol., durab./ Dependab.:availab., reliability	Performance. Running examp./ None
Mukhija and Glinz CASA [18]	QoS preser. Self-config.	Contracts: QoS	SLOs/Logic. properties of comput. elem. ext. context	Discrete oper. managed system's soft. architecture	Reconfig. ctrl./Modif. structure reflection	Consistency/ Performance	Performance. Running examp./ None
Parekh et al. [20]	QoS preser.	Single reference value	SLOs/Logic. properties of comput. elem.	Continuous signals behavioral properties	Feedback ctrl./ Non-modif.struct. math.model	Stab., overshoot/ Performance: throughput, capacity	Lotus Notes Running examp./ Offered load
Sicard et al. [23]	Self-manag. self-healing	Constraints: comput. states	Not explicit monitoring	Discrete oper. managed system's soft. architecture	Feedback ctrl./Modif. structure reflection	None for the controller/ Dependab.: reliab., avail.	Performance. Simulations/ Availability $A = MTTR$
Solomon et al. [24]	Self-optim.	Contracts: QoS	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's soft. architecture	Adaptive ctrl./Modif. structure reflection	Accuracy/ Performance	Accuracy. Running examp./ None
Tamura et al. SCe-SAME [25]	QoS preser.	Contracts: SLOs-QoS	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's soft. architecture	Reconfig. ctrl./Modif. struct.reflect.	Termination, consistency/ QoS prop., dependab.: reliability	Formal verif. of properties. Running examp./ None
White et al. Autonomic JBeans [27]	Self-manag. Self-healing Self-protect.	Contracts: SLOs-QoS	SLOs/Logic. properties of comput. elem.	Discrete oper. managed system's process	Adaptive ctrl./Non-modifiable structure	Settl. time/ Perform.:through., Depend.: availab.	Empirical. Develop. effort/ Average response time

take the external context into account [14, 18]. Regarding the controller structure, all approaches, except [20] and [23] that implement a simple feedback loop, implement either adaptive or reconfigurable control. Finally, the most common evaluation mechanism is the implementation of running examples based on simulated environments. Table 3 summarizes the characterization and classification of the studied self-adaptive approaches presented in Table 2.

Table 2: Characterization Summary

Characteristic	Count [List of Approaches]
Spectrum Classification	
Control Engineering	1 [20]
Hybrid	5 [3, 4, 5, 8, 27]
Hybrid-Software	1 [24]
Software Engineering	9 [1, 6, 9, 10, 14, 15, 18, 23, 25]
Monitoring Mechanisms	
Monitor internal context	15
Monitor external context	2 [14, 18]
Non specified	1 [23]
Controller's Structure	
Feedback control	2 [20, 23]
Adaptive control	9 [1, 3, 4, 8, 9, 10, 14, 24, 27]
Reconfigurable Control	4 [5, 6, 15, 18, 25]
Managed System's Structure	
Non-modifiable	4 [3, 8, 20, 27]
Modifiable with reflection	12 [1, 4, 5, 6, 9, 10, 14, 15, 18, 23, 24, 25]
Adaptation Properties	
Settling time	4 [1, 4, 14, 27]
Small overshoot	4 [1, 4, 14, 20]
Scalability	3 [1, 6, 9]
Stability	2 [1, 20]
Accuracy	2 [5, 24]
Termination	2 [8, 25]
Consistency	3 [15, 18, 25]
Robustness	1 [6]
Security	0

4. MEASURING ADAPTATION PROPERTIES

Our evaluation of a self-adaptive system has two aspects. The first one concerns the evaluation of desired properties for the managed system. In our analysis we focused only on desired properties that correspond to quality attributes of software systems. The second one relates to desired properties for the controller for the adaptation process. For the identification of desired properties for the managed system, we based our analysis on the taxonomy of quality attributes for software systems proposed by SEI researchers [2]. For properties related to the controller, we based our analysis on the *SASO properties* identified by Hellerstein et al. in the application of control theory to computing systems [11], and other properties identified in self-adaptive software systems surveys [8, 15, 17, 18]. Furthermore, we classified the identified adaptation properties according to *how* and *where* they are observed (cf. Table 4.2). Concerning how they are observed, some properties can be evaluated using static verification techniques while others require dynamic verification and run-time monitoring. We use the term *observed* as some properties are difficult to measure, despite the fact that controllers are designed to preserve them [25]. With respect to where, properties can be evaluated on the managed system or on the controller. On the one hand, some properties to evaluate the controller are observable on the controller itself or on both the controller and the managed system; however, most properties can only be observed on the managed system. On the other hand, properties to evaluate the managed system are observable only on the managed system. In both cases, the environment that can affect the behavior of the controller or the managed system is also a factor worth of consideration.

Based on Sect. 2 and the analysis presented in Sect. 3, this section presents the foundations of a framework composed of a set of properties and metrics for the evaluation of self-adaptation, where properties observable on the managed system, either to evaluate the controller or the managed system, can be evaluated in terms of quality attributes. As part of our framework, we propose a process for evaluating self-adaptation with which software engineers should be able to (i) identify required adaptation goals (i.e., the quality attributes that drive the adaptation of the managed system); (ii) identify adaptation properties to evaluate the controller, including the properties that are observable on the controller itself, on the managed system, or on both; (iii) map quality attributes used to evaluate the managed system to properties that evaluate the controller but are observable on the managed system; and (iv) define metrics to evaluate properties observable on the managed system and on the controller.

4.1 Quality Attributes as Adaptation Goals

If we intend to evaluate an adaptive software system we need to identify the motivation behind building it—the *adaptation goal*. In general, adaptation can be motivated by the need of continued satisfaction of functional and regulation of non-functional requirements under changing context conditions. Nevertheless, as most analyzed contributions focus on non-functional factors, we based our analysis on software systems whose adaptation goals are motivated by quality concerns. Moreover, characteristics of self-adaptive systems, such as self-configuring or self-optimizing, can be mapped to quality attributes. Following this idea, Salehie and Tahvildari discussed the relationships between autonomic characteristics and quality factors such as the relationship between self-healing and reliability [22]. Our main contribution in this paper is the application of quality attributes to evaluate self-adaptive software systems, as quality attributes are commonly used to evaluate desirable properties on the managed system. More importantly, we propose a mapping between quality factors and adaptation properties. In fact, this introduces a level of indirection for evaluating adaptation properties that are not directly observable on the controller. In this subsection we present the definitions of the selected quality attributes introduced in Sect. 2, with corresponding citations of the analyzed contributions that address them.

Performance. Characterizes the timeliness of services delivered by the system. It refers to responsiveness, that is, the time required for the system to respond to events or the event processing rate in an interval of time. Identified factors that affect performance are latency (the time the system takes to respond to a specific event [5, 10, 18]); throughput

(the number of events that can be completed in a given time interval—beyond processing rate as the desired throughput must also be observed in time sub-intervals [6, 14, 20, 24, 27]); and capacity (a measure of the amount of work the system can perform [6, 14, 20]).

Dependability. Defines the level of reliance that can justifiably be placed on the services the software system delivers. Adaptation goals related to dependability are availability (readiness for usage [1, 3, 4, 15, 23, 27]); reliability (continuity of service [3, 8, 9, 14, 15, 23]); maintainability (capacity to self-repair and evolve [1, 4, 9, 23]); safety (from a dependability point of view, non-occurrence of catastrophic consequences from an external perspective (on the environment) [3]); confidentiality (immune to unauthorized disclosure of information); integrity (non-improper alterations of the system structure, data and behavior [3]).

Security. The selected concerns of the security attribute are confidentiality (protection from disclosure); integrity (protection from unauthorized modification); and availability (protection from destruction [3]).

Safety. The level of reliance that can justifiably be placed on the software system as not generator of accidents. Safety is concerned with the occurrence of accidents, defined in terms of external consequences. The taxonomy presented in [2] includes two properties of critical systems that can be used as indicators of system safety: interaction complexity and coupling strength. In particular, interaction complexity is the extent to which the behavior of one component can affect the behavior of other components. SEI's taxonomy presents detailed definitions and indicators for these two properties.

4.2 Adaptation Properties

A second part of our contribution is the identification of adaptation properties that have been used for the analyzed spectrum of adaptive systems, from control theory to software engineering, to evaluate the adaptation process. The identified adaptation properties are stated as follows. The first four properties, called *SASO properties,* correspond to desired properties of controllers from a control theory perspective [11]; note that the stability property has been widely applied in adaptation control from a software engineering perspective. The remaining properties in the list were identified from hybrid approaches. Citations included in each property definition refer either to papers where the property was also defined or to examples of adaptive systems where the property is observed in the adaptation process.

Stability. The degree in that the adaptation process will converge toward the control objective. An unstable adaptation will indefinitely repeat the controlling action with the risk of not improving or even degrade the managed system to unacceptable or dangerous levels. In a stable system, responses to a bounded input are bounded to a desirable range [1, 9, 16, 17, 20].

Accuracy. This property is essential to ensure that adaptation goals are met, within given tolerances. Accuracy must be measured in terms of how close the managed system approximates to the desired state (e.g., reference input values for quality attributes) [5, 24].

Short settling time. The time required for the adaptive system to achieve the desired state. The settling time represents how fast the system adapts or reaches the desired state. Long settling times can bring the system to unstable states. This property is commonly referred to as recovery time, reaction time, or healing time [4, 11, 14, 16, 17].

Small overshoot. The utilization of computational resources during the adaptation process to achieve the adaptation goal. Managing resource overshoot is important to avoid the system unstability. This property expresses how well the adaptation performs under given conditions—the amount of resources used in excess to achieve a required short settling-time before reaching a stable state [1, 4, 14, 16, 20].

Robustness. The managed system must remain stable and guarantee accuracy, short settling time, and small overshoot even if the managed system state differs from the expected state in some measured way. Also, the adaptation process is robust if the controller is able to operate within desired limits even under unforeseen conditions [6, 17].

Termination (of the adaptation process). In software engineering approaches, the planner in the MAPE-K loop typically produces discrete controlling actions to adapt the managed system (cf. Sect. 2), such as a list of component-based architecture operations. The termination property guarantees that this list is finite and its execution will finish, even if the system does not reach the desired state. Termination is also referred as deadlock-free execution, meaning that, for instance, a reconfigurable adaptation process must avoid adaptation rules with deadlocks among them [8, 25].

Consistency. This property aims at ensuring the structural and behavioral integrity of the managed system after performing an adaptation process. For instance, when a controller's adaptation plan is based on dynamic reconfiguration of software architecture, consistency concerns are to guarantee sound interface bindings between components (e.g., component-based structural compliance) and to ensure that when a component is replaced dynamically by another one, the execution must continue without affecting the function of the affected component. These concerns help protect the application from reaching inconsistent states as a result of dynamic recomposition [18]. Léger et al. define this property alongside atomicity, isolation and durability to complete the *ACID* properties found in transactional systems for guaranteeing reliability in transactions [15]: *(i) Atomicity*, either the system is adapted and the adaptation process finishes successfully or it is not finished and the adaptation process aborts. If an adaptation process fails, the system is returned to its previous consistent state; *(ii) isolation*, adaptation processes are executed as if they were independent. Results of unfinished adaptation processes are not visible to others until the process finishes. Results of aborted or failed adaptation processes are discarded; and *(iii) durability*, the results of a finished adaptation process are permanent: once an adaptation process finishes successfully, the new system state is made persistent. In case of major failures (e.g., hardware failures), the system state can be recovered.

Scalability. The capability of a controller to support increasing demands of work with sustained performance using additional computing resources. For instance, scalability is an important property for the controller when it must evaluate an increased number of conditions in the analysis of context. As computational efficiency is relevant for guaranteeing performance properties in the controller, controllers are required to avoid the degradation of any of the operations of the adaptive process in any situation [1, 6, 9].

Security. In a secure adaptation process, not only the target system but also the data and components shared with the controller are required to be protected from disclosure (confidentiality), modification (integrity), and destruction (availability) [2].

Table 3: Classification of adaptation properties

Adaptation Property	Property Verification Mechanism	Where the Property is Observed
Stability	Dynamic	Managed system
Accuracy	Dynamic	Managed system
Settling Time	Dynamic	Managed system
Small Overshoot	Dynamic	Managed system
Robustness	Dynamic	Controller
Termination	Static	Controller
Consistency	Both	Managed system
Scalability	Dynamic	Both
Security	Dynamic	Both

4.3 Mapping Adaptation Properties to Quality Attributes

Once the adaptation goal and adaptation properties have been identified, the following step maps the properties of the controller, which are observable on the managed system, to quality attributes on the managed system. Table 4.3 presents a general mapping between adaptation properties and quality attributes. These quality attributes refer to attributes on both the controller and the managed system depending on where the corresponding adaptation properties are observed. According to Tables 4.2 and 4.3, SASO properties (i.e., stability, accuracy, settling time and small overshoot) can be verified at run-time by observing performance, dependability and security factors in the managed system.

With respect to *stability*, Océano, the dynamic resource allocation system that supports SLAs for peak loads with an order of magnitude of difference, addresses stability based on dependability (i.e., availability and maintainability), and performance (i.e., throughput and capacity—scalability) [1]. In a similar way, the controller proposed by Parekh et al. to guarantee desirable performance levels (i.e., throughput and capacity) also addresses stability as an adaptation property [20]. They applied an integral control technique to construct a transfer function that models the system and the way the behavior of the managed system is affected by the controller. Baresi and Guinea also addressed stability by proposing a self-recovery system where service oriented architecture (SOA) business processes recover from disruptions of functional and non-functional requirements to avoid catastrophic events (safety) and improper system state alterations (integrity), guaranteeing readiness for service (availability) and correctness of service (reliability) [3].

Concerning *accuracy*, the MOSES framework proposed by Cardellini et al. uses adaptation policies in the form of directives to select the best implementation of the composite service according to a given scenario [5]. MOSES adapts chains of service compositions based on service selection using a multiple service strategy. It has been tested with multiple adaptation goals to observe the behavior of the adaptation strategy in terms of its accuracy (i.e., how close the managed system reaches the adaptation goal). Solomon et al. also address accuracy in their self-optimizing mechanism for business processes [24]. They used a simulation model to anticipate performance levels and make decisions about the adaptation process. For this, a tuning algorithm keeps the simulation model accurate. This algorithm compensates for the measurement of actual service time to increase the accuracy of simulations by modeling errors, probabilities and inter-arrival times. Then, it obtains the best estimate for these data such that the square root of the difference between the simulated and measured metrics is minimized.

Table 4: Mapping properties to quality attributes

Adaptation Property	Quality Attributes	
Stability	Performance	Latency Throughput Capacity
	Dependability	Safety Integrity
	Security	Integrity
Accuracy	Performance	Latency Throughput Capacity
Settling Time	Performance	Latency Throughput
Small Overshoot	Performance	Latency Throughput Capacity
Robustness	Dependability	Availability Reliability
	Safety	Interact. Complex. Coupling Strength
Termination	Dependability	Reliability Integrity
Consistency	Dependability	Maintainability Integrity
Scalability	Performance	Latency Throughput Capacity
Security	Security	Confidentiality Integrity Availability

Regarding *settling time*, Appleby et al measure settling time in terms of the time required for deploying a new processing node including the installation and reconfiguration of all applications and data repositories in Océano [1]. Similarly, White et al. evaluated settling time in terms of the average response time required for autonomic EJBs to adapt [27]. Candea's approach, on another side, applied a recursive strategy that reduces mean time to repair (MTTR) by means of recovering minimal subsets of failed system components. If localized and minimal recovery is not enough, progressively larger subsets are recovered [4].

With respect to the last SASO property, *small overshoot*, the control-based approach to ensure SLOs proposed by Parekh et al. addresses it by avoiding that control values (e.g., MAXUSERS) are set to values that exceed their legal range. They used root-locus analysis to predict the valid values of the maximum number of users. This approach divides the valid range of values into three regions in order

to decide when the control values reach undesirable levels. Based on empirical studies, they analyzed properties of the transfer function to predict the desired range of values [20].

Robustness is addressed in the self-management approach for balancing system load proposed by Dowling and Cahill [6]. They aimed at realizing a robust controller by implementing an adaptation mechanism via decentralized agents that eliminate centralized points of failure.

Termination can be verified using dynamic and static mechanisms. Ehrig et al. proposed a self-healing mechanism for a traffic light system to guarantee continuity of service (reliability) by self-recovering from predicted failures (integrity) [8]. They addressed termination by statically checking that self-healing rules are deadlock-free, in such a way that the self-repairing mechanism never inter-blocks traffic lights in the same road intersection.

Scalability is also an adaptive property in the K-Components system [6]. For this, self-management local rules of the agents support evolving capabilities. Another approach where scalability is addressed as an adaptation property is Madam, the middleware proposed by Floch et al. for enabling model-based adaptation in mobile applications [9]. Scalability is a concern in Madam for several reasons. First, its reasoning approach might result in a combinatorial explosion if all possible variants are evaluated; second, the performance of the system might be affected when reasoning on a set of a concurrently running applications competing for the same set of resources. They proposed a controller where each component (e.g., the adaptation manager) can be replaced at run-time to experiment with different analysis approaches for managing scalability.

Security was not addressed as an adaptation property by any of the self-adaptive systems analyzed. However, we propose the use of SEI's definition of security as a quality attribute and its corresponding quality factors to evaluate security on the controller [2]. As presented in Table 4.2, security of the controller should be evaluated independently of the managed system. This means that ensuring security at the managed system does not guarantee security in the controller.

4.4 Adaptation Metrics

Adaptation metrics provide the way of evaluating adaptive systems with respect to particular concerns of the adaptation process [17, 21]. Thus, metrics provide a measure to evaluate desirable properties. For instance, metrics to evaluate control systems measure aspects concerning the SASO properties (i.e., stability, accuracy, settling time, and small overshoot). To characterize the evaluation of adaptive systems, we analyzed the variety of self-adaptive software systems to identify adaptation properties (i.e., at the managed system and the controller) that were evaluated in terms of quality attributes (cf. Sects. 3 and 4.1). Just as evaluating most properties is impossible by observing the controller itself, we propose the evaluation of these properties by means of observing quality attributes on the managed system. To identify relevant metrics, we characterized a set of factors that affect the evaluation of quality attributes such as memory usage, throughput, response time, processing rate, mean time to failure, and mean time to repair [16, 2]. These factors are essential when considering the metrics to evaluate properties on both the controller on the managed system [21].

Cardellini et al. evaluated performance and reliability in MOSES using the following metrics: expected response time (Ru) (the average time needed to fulfill a request for a composite service); expected execution cost (Cu) (the average price to be paid for a user invocation of the composite service); and expected reliability (Du) (the logarithm of the probability that the composite service completes its task for a user request [5]).

Appleby et al. measured dependability factors (e.g., availability) and performance factors (e.g., scalability in terms of throughput and capacity) in Océano using the following metrics: active connections per server (the average number of active connections per normalized server across a domain); overall response time (the average time it takes for any request to a given domain to be processed); output bandwidth (the average number of outbound bytes per second per normalized server for a given domain); database response time (average time it takes for any request to a given domain to be processed by the back-end database); throttle rate (T) (a percentage of connections disallowed to pass through Océano on a customer domain); admission rate (the complement of the domain throttle rate ($1 - T$)); and active servers (the number of active normalized-servers which service a given customer domain [1]).

Average response time is a common metric used to evaluate performance in several adaptive approaches, such as the framework to develop autonomic EJB applications proposed by White et al. [27]. Another example is K-Components, which optimizes system performance based on a load balancing function on every adaptation contract that uses a cost function to calculate its internal load cost and the ability of its neighbors to handle the load [6]. This cost function is defined as the addition of the advertised load cost and internal cost of the component (i.e., calculated as the estimated cost to handle a particular load type).

Parekh et al., in their control-based approach to achieve performance service level objectives, used the length of the queue of the in-progress client requests as the metric to control the offered load (the load imposed on the server by client requests) [20]. Baresi and Guinea also proposed a metric to control reliability on the adaptation of BPEL business processes [3], based on the number of times a specific method responds to within two minutes over the total number of invocations. Moreover, they defined a KPI based on this metric, such that reliability must be greater that 95% over the past two hours of operation.

Kumar et al. defined a business value KPI in terms of factors, such as the priority of the user accessing the information, the time of day the information is being accessed, and other aspects that determine how critical the information is to the enterprise [14]. For this, they used a utility function as a combination of some of these factors: $utility(e_{gj-k}) = f(\sum d_{ni}, min(b_n i), b_{gj-k})$, where $i|e_{ni} \in M(e_{gj-k})$. The business utility of each edge (e_{gj-k}), which represents data streams between operators that perform data transformations, is a function on the delay d_{ni}, the available bandwidth $b_n i$ of the intervening network edges e_{ni}, and the required bandwidth b_{gj-k} of the edge e_{gj-k}.

Candea et al. evaluated availability in terms of mean time to recover ($MTTR$) [4]. For this, they defined two metrics: availability ($A = MTTF/(MTTF + MTTR)$) and downtime of unavailability ($U = MTTR/(MTTF + MTTR)$), where $MTTF$ is the mean time for a system or subsystem to fail (i.e., the reciprocal of reliability), $MTTR$ is the mean

time to recover, and A is a number between 0 and 1. U can be approximated to $MTTR/MTTF$ when $MTTF$ is much larger than $MTTR$. Similarly, Sicard et al. defined a metric for availability in terms of $MTTR$ [23].

Last column of Table 2 summarizes the identified evaluation methods and metrics to assess self-adaptive systems. Although these metrics are directly related to the measurement of quality factors, we expect that these metrics will be useful for evaluating adaptation properties based on our proposed mapping between quality attributes and adaptation properties (cf. Sect. 4). The approach by Reinecke et al. [21] supports our hypothesis. Their metric measures the ability of a self-adaptive system to adapt. They argue that adaptivity can be evaluated using a meta-metric named *payoff* which can be defined in terms of any performance metric to measure the effectiveness of the adaptation process. That is, the optimal adaptive system is characterized by the fact that its adaptation decisions are always optimal (i.e., always yield the optimal payoff). To apply their metric it is necessary to (i) identify the adaptation tasks, (ii) define one or more performance metrics on these tasks (i.e., these metrics should reflect the contribution of these tasks toward the adaptation goal), (iii) define a payoff metric in terms of the performance metrics, and (iv) to apply the metric by observing it in the system's performance.

5. DISCUSSION

We started the analysis phase for this work with 34 research papers with different proposals for self-adaptive software systems published over the past decade. From this set, 18 were filtered-out mainly because either they presented very generic proposals (i.e., with non-measurable self-adaptive properties) or they did not include enough information in the paper for characterization purposes. From the analysis of the 16 remaining papers that we presented in the previous sections (and even considering the 18 papers that we filtered out), it is worth noting that the prevalent difficulty—or the lack of awareness of—to identify metrics to evaluate self-adaptive software. Nonetheless, some advances have been made based on concepts from control theory and the recognized importance of quality metrics and corresponding measurements as the basis for understanding and improving adaptive processes. However, as we can conclude from our analysis, there are plenty of opportunities and challenges to be addressed, even when we consider concepts more abstract than metrics, such as self-adaptation properties. In the following we outline some of these opportunities and challenges.

First, most of the proposals focus only on self-adaptation mechanisms, not addressing explicitly the level of achievement of adaptation properties nor the adaptation properties themselves. On the one hand, it is known that even though control theory has defined standardized properties that a controller must realize (i.e., the SASO properties), self-adaptive software systems require additional properties, due to their discrete nature (e.g., termination). On the other hand, it is clear from the discussion in previous sections that quality attributes are a plausible option to evaluate some adaptation properties. However, it will be necessary (i) to evaluate whether our proposed set of self-adaptation properties defined in Sect. 4.2 is general enough to assess self-adaptation mechanisms; (ii) to find standardized metrics to measure self-adaptation properties, which could be based on proposals such as the one by Reinecke et al. [21]; and (iii) to analyze whether the proposal of measuring adaptation properties based on quality attributes is meaningful enough, and if they fulfill the conceptual definitions of corresponding properties of, for instance, control theory. Moreover, our framework must be extended to support trade-off analysis when multiple adaptation properties and quality attributes are involved.

Second, the lack of awareness of adaptation properties as a goal to be measured results in a lack of evaluation methods and metrics for these properties and for the adaptation mechanisms themselves. However, this trend could be reversed by designing self-adaptive mechanisms with implicit controllable and measurable properties. For some verifiable properties, this can be obtained by developing or using formal models as a basis for self-adaptation processes. For adaptation mechanisms with measurable properties, a main challenge is to develop mathematical behavior models based on the architecture itself of the target computing systems to be controlled.

Third, without declared evaluation methods and metrics it is very difficult to compare and reason about the engineering of self-adaptation; for instance, from our analysis it was impossible to identify any measurable relationship between the adaptation goals and the evaluation of the adaptation strategies as such. From the structural point of view, it is clear that decoupling the controller from the managed system, with respect to evaluation, is a first critical step toward controlling the properties of dynamic self-adaptation. However, several questions remain: Does the system structure (i.e., controller and managed system) have any significant relationship with the quality of the adaptation approach? Do non-explicit controllers imply non-explicit or even the absence of adaptation properties? From the behavior point of view and considering the adaptation mechanism as a black-box, how do we compare managed system behavior in the different phases of the adaptation process? Maybe in terms of stability and other properties such as settling time. Finally, under which circumstances and characteristics of the managed system is an adaptation mechanism better than another? There is no available evaluation framework to compare adaptation mechanisms to help answer these questions.

6. CONCLUSIONS

Self-adaptive software evaluates and modifies its behavior to preserve the satisfaction of its functional requirements and regulation of non-functional requirements, under changing context conditions of execution. Most importantly, even though researchers devised and proposed many diverse and valuable strategies to modify the behavior of a managed system, many of these contributions did not identify nor address adaptation properties and corresponding metrics. Thus, adaptive systems are generally not evaluated explicitly —neither in the controller nor in the managed system. Hence, validation mechanisms discussed for these approaches are usually limited to the evaluation of performance properties observed in the managed system, even when the adaptation goal is not related to performance quality attributes. Future work will focus on the validation of the adaptation properties and their mapping to quality attributes as proposed in this paper through evaluation of existing adaptive systems.

Acknowledgments

This work was funded in part by University of Victoria (Canada), the National Sciences and Engineering Research Council (NSERC) of Canada, IBM Corporation, Icesi University and University of Los Andes (Colombia), and Ministry of Higher Education and Research of Nord-Pas de Calais Regional Council and FEDER under Contrat de Projets Etat Region (CPER) 2007-2013.

7. REFERENCES

[1] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. P. Pazel, J. Pershing, and B. Rochwerger. Océano - SLA based management of a computing utility. In *7th IFIP/IEEE International Symposium on Integrated Network Management*, pages 855–868, 2001.

[2] M. Barbacci, M. H. Klein, T. A. Longstaff, and C. B. Weinstock. Quality attributes. Technical Report CMU/SEI-95-TR-021, CMU/SEI, 1995.

[3] L. Baresi and S. Guinea. Self-supervising BPEL processes. *IEEE Transactions on Software Engineering*, 99(PrePrints), 2010.

[4] G. Candea, J. Cutler, and A. Fox. Improving availability with recursive microreboots: a soft-state system case study. *Performance Evaluation*, 56(1-4):213 – 248, 2004. Dependable Systems and Networks - Performance and Dependability Symposium (DSN-PDS) 2002: Selected Papers.

[5] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola. QoS-driven runtime adaptation of service oriented architectures. In *ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 131–140, New York, NY, USA, 2009. ACM.

[6] J. Dowling and V. Cahill. Self-managed decentralised systems using K-components and collaborative reinforcement learning. In *1st ACM SIGSOFT Workshop on Self-Managed Systems*, pages 39–43, New York, NY, USA, 2004. ACM.

[7] G. Dumont and M. Huzmezan. Concepts, methods and techniques in adaptive control. In *the 2002 American Control Conference*, volume 2, pages 1137 – 1150. IEEE, 2002.

[8] H. Ehrig, C. Ermel, O. Runge, A. Bucchiarone, and P. Pelliccione. Formal analysis and verification of self-healing systems. In *Fundamental Approaches to Software Engineering*, volume 6013 of *LNCS*, pages 139–153. Springer, 2010.

[9] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. Using architecture models for runtime adaptability. *IEEE Software*, 23:62–70, 2006.

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

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

[12] J. L. Hellerstein, S. Singhal, and Q. Wang. Research challenges in control engineering of computing systems. *IEEE Transactions on Network and Service Management*, 6(4):206–211, 2009.

[13] IBM Corporation. An architectural blueprint for autonomic computing. Technical report, IBM Corporation, 2006.

[14] V. Kumar, B. F. Cooper, Z. Cai, G. Eisenhauer, and K. Schwan. Middleware for enterprise scale data stream management using utility-driven self-adaptive information flows. *Cluster Computing*, 10:443–455, 2007.

[15] M. Léger, T. Ledoux, and T. Coupaye. Reliable dynamic reconfigurations in a reflective component model. In *13th International Symposium on Component Based Software Engineering*, volume 6092 of *LNCS*, pages 74–92. Springer, 2010.

[16] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance specifications and metrics for adaptive real-time systems. In *Real-Time Systems Symposium*, 2000.

[17] A. C. Meng. On evaluating self-adaptive software. In *1st International Workshop on Self-Adaptive Software*, pages 65–74, Secaucus, NJ, USA, 2000. Springer.

[18] A. Mukhija and M. Glinz. Runtime adaptation of applications through dynamic recomposition of components. In *18th International Conference on Architecture of Computing Systems*, 2005.

[19] K. S. Narendra and J. Balakrishnan. Adaptive control using multiple models. *IEEE Transactions on Automatic Control*, 42:171–187, 1997.

[20] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Systems*, 23:127–141, 2002.

[21] P. Reinecke, K. Wolter, and A. van Moorsel. Evaluating the adaptivity of computing systems. *Performance Evaluation*, 67(8):676 – 693, 2010. Special Issue on Software and Performance.

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

[23] S. Sicard, F. Boyer, and N. De Palma. Using components for architecture-based management: the self-repair case. In *30th International Conference on Software Engineering*, pages 101–110. ACM, 2008.

[24] A. Solomon, M. Litoiu, J. Benayon, and A. Lau. Business process adaptation on a tracked simulation model. In *2010 Conference of the Center for Advanced Studies on Collaborative Research*. ACM, 2010.

[25] G. Tamura, R. Casallas, A. Cleve, and L. Duchien. QoS contract-aware reconfiguration of component architectures using e-graphs. In *7th International Workshop on Formal Aspects of Component Software*, LNCS. Springer, 2010.

[26] United States Air Force Chief Scientist (AF/ST). Technology Horizons a Vision for Air Force Science & Technology During 2010-2030. Technical report, U.S. Air Force, 2010.

[27] J. White. Simplifying autonomic enterprise Java Bean applications via model-driven development: a case study. *The Journal of Software and System Modeling*, pages 601–615, 2005.