ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/295072411

A Framework for the Generation and Management of Self-Adaptive Enterprise Applications

Conference Paper · September 2015

DOI: 10.1109/ColumbianCC.2015.7333412

CITATIONS
3

READS
44

4 authors, including:

Hugo Arboleda
University ICESI

50 PUBLICATIONS **137** CITATIONS

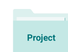
SEE PROFILE

Gabriel Tamura
University ICESI

48 PUBLICATIONS **601** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



SHIFT: A Framework for the Generation and Management of Self-Adaptive Enterprise Applications View project

AMPLE -- Aspect Oriented, Model Driven, Software Product Lines View project

All content following this page was uploaded by Andrés Felipe Paz on 19 February 2016.

The user has requested enhancement of the downloaded file.

A Framework for the Generation and Management of Self-Adaptive Enterprise Applications

Hugo Arboleda*, Andrés Paz†, Miguel Jiménez‡ and Gabriel Tamura§
Universidad Icesi, I2T Research Group
Cali, Colombia
{ *hfarboleda, †afpaz, ‡majimenez, §gtamura }@icesi.edu.co

***Abstract*—Demand for self-adaptive enterprise applications has been on the rise over the last years. Such applications are expected to satisfy context-dependent quality requirements in varying execution conditions. Their dynamic nature constitutes challenges with respect to their architectural design and development, and the guarantee of the agreed quality scenarios at runtime. In this paper we present the constituting elements of SHIFT, a framework that integrates (i) facilities and mechanisms for managing self-adaptive enterprise applications, (ii) automated derivation of self-adaptive enterprise applications and their respective monitoring infrastructure, and (iii) decision support for the assisted recomposition of self-adaptive applications.**

***Index Terms*—Self-adaptive enterprise applications, software product lines, component configurations.**

I. INTRODUCTION

Enterprise applications (EAs) are intended to satisfy the needs of entire organizations and usually involve persistent data, concurrent user access to the information and several user interfaces to handle the big amount of data requested. They live in dynamic execution contexts and are no longer isolated but instead interacting with other systems. Their dynamic nature implies that they are constantly under the influence of external stimuli (*i.e.* disturbances) from various sources inside or outside the system scope that may affect their behaviour or the levels at which they satisfy agreed quality. Regardless of the intrinsic uncertainty of disturbances and their possible sources, EAs still have to fulfill the customers' quality agreements. This has generated a growing interest regarding support of infrastructures for autonomic adaptation of EAs, as well as flexible architectural designs conceived for allowing recomposition at runtime.

In this paper we present preliminary results regarding our SHIFT framework, which provides (i) facilities and mechanisms for managing self-adaptive enterprise applications based on the adaptation feedback loop of the DYNAMICO reference model [1], (ii) support for automated derivation of self-adaptive enterprise applications considering possible functional, quality and monitoring variations, and (iii) automated reasoning at runtime regarding context- and system-sensed data to determine and apply necessary system adaptations, considering deployment and undeployment tasks. SHIFT's constituent elements are at different stages of development; throughout this paper we specify their current states.

The remainder of this paper is organized as follows. Section II introduces the background and motivation. Section III provides a general description of the SHIFT framework. Section IV presents the mechanisms for assisted derivation of applications. Section V describes our concrete implementation of the architecture for managing self-adaptive enterprise applications. Section VI presents our adaptation planning strategy based on automated reasoning. Section VII sets out conclusions and outlines future work.

II. BACKGROUND AND MOTIVATION

Current approaches implement dynamic adaptation of service compositions at the language level [2], [3], [4], or using models at runtime [5], [6], [7], [8]. The first ones can be complex and time-consuming, and with low-level implementation mechanisms. Our work is related to approaches that use models at runtime. Model-based approaches for dynamic adaptation implement, tacit or explicitly, the MAPE-K reference model [9] that comprises five elements: (i) a Monitor, (ii) an Analyzer, (iii) a Planner, (iv) an Executor, and (v) a Knowledge base.

The recent work of Alferez *et al.* [10] summarizes good practices implementing the MAPE-K reference model and gives implementation details about reconfiguration mechanisms. They center their attention on service recomposition at runtime using (dynamic) product line engineering practices for assembling and redeploying complete applications according to context- and system-sensed data. Model-based approaches for dynamic adaptation of service compositions (*e.g.*, [10], [5], [11] do not consider changing requirements over SCA composites or EJB models. This triggers new challenges given the complexity of deployment at the stage of adapting composites and EJB bindings. The work of van Hoorn *et al.* [12] gives an adaptation framework operating over component-based software systems. Their proposal remains at a high level without working with specific component models. Their framework is centered around component migration and load balancing, while our interest is component recomposition.

The work of Cedillo *et al.* in [5] is also closely related to ours. They propose a middleware for monitoring cloud services defined around a monitoring process that uses models at runtime capturing low- and high-level non-functional requirements from Service Level Agreements (SLAs). Their middleware only provides a partial implementation of the MAPE-K reference model, specifically of the monitor and analyzer elements. Their proposal derives the monitoring code

from the input model at runtime. The monitoring code is used by the middleware during the monitoring process. Heinrich *et al.* [11] also work around monitoring cloud applications. however, they are only concerned with triggering change events when the observation data model is populated at runtime.

Other approaches lack support for assisted derivation of monitoring infrastructures, which is important in order to manage reference architectures that prevent the infrastructure from introducing considerable overhead in the system's regular operations. Assisted derivation of monitoring infrastructure also guarantees relevance of the complete self-adaptive architecture in changing context conditions of system execution [1].

In previous work, we proposed independent approaches and implementations in the contexts of the engineering of highly dynamic adaptive software systems with the DYNAMICO reference model [1], quality of service (QoS) contract preservation under changing execution conditions with the QoS-CARE implementation [13], model-based product line engineering with the FIESTA approach [14], [15], [16], automated reasoning for derivation of product lines [17], and the recent (unpublished) contributions regarding quality variations in the automated derivation process of product lines [16]. The SHIFT framework is motivated by the required integration of all these efforts in a move to approach automation and quality awareness along the life cycle of enterprise applications. With SHIFT we are currently focused in the design, development, deployment and operation stages of the life cycle. The remaining stages (*e.g.*, testing, maintenance/evolution) are part of our ongoing research work.

III. Framework Overview

Figure 1 presents a high-level architectural view of SHIFT's constituting elements and their data flow. They are grouped into 2 areas: `Automated Derivation` and `Autonomic Infrastructure`. The `Automated Derivation` area is concerned about providing support for functional and quality configuration and derivation of (i) deployable enterprise applications components and (ii) monitoring infrastructure. In this area we use a model to capture reference architectures (`Reference Architecture` submodel) built on design patterns and their composition, and the functional (`Domain` submodel), quality (`Decision Support` submodel) and monitoring (`Monitoring Infrastructure` submodel) scopes of the EAs. Generated component sets and quality decision models, relating component sets with quality scenarios, are stored in the `Component and Quality Repository`, which is managed by the `Knowledge Manager` element; they are an input for (re)deployment processes.

The monitoring infrastructure is deployed as part of the `Autonomic Infrastructure` area, which implements the adaptation feedback loop of the DYNAMICO reference model [1]. As part of the `Autonomic Infrastructure` area, SHIFT considers the need for dynamically deploying and undeploying components to realize adaptation plans. Thus, the `Planner` element has to provide automated reasoning on the dynamic creation of structural adaptation plans. In order to obtain the best possible selection of components when configuring an adaptation to a deployed product, we rely on constraint satisfaction to reason on the set of constraints defined by reachable quality scenarios configurations and their relationships with the component sets implementing them. Interactions between quality scenarios may occur, and since different component sets may be available, conflicts between component sets may arise. Through automated reasoning, the `Planner` element may cope with this issue by taking into account additional information to get the best possible selection of component sets when determining an adaptation plan.

Realizing an adaptation plan in the `Executor` element considers transporting components from their source repository to the corresponding computational resource, undeploying previous versions of them, deploying them into the middleware or application server, binding their dependencies and services, and executing them. All of these while redirecting new requests for the application's components to the new instances being deployed, and allowing existing requests and sessions to terminate. In addition, if necessary, to recompile system source code to make measurement interfaces available to the monitoring infrastructure. Accordingly, these deployment tasks are applied to the `Monitor` element to effectively ensure dynamic quality awareness.

IV. Automated Derivation of Applications and Monitors

The `Automated Derivation` area in Figure 1 contains a model comprised of four submodels: `Reference Architecture`, `Domain`, `Decision Support` and `Monitoring Infrastructure`. We have currently devised the first three submodels (*i.e.* `Reference Architecture`, `Domain`, `Decision Support`), which we detail in Figure 2 with a UML-like notation. Subsection IV-A explains the three submodels we have specified so far. Subsection IV-C outlines how we will address the `Monitoring Infrastructure` submodel.

A. Specification and Design of Functionality and Quality

Functional Scope. The domain submodel (see Figure 2 left bottom corner) comprises an extensible metamodel for capturing the functional scope of product lines in the context of enterprise applications. This is based on our previous work [16]. The metamodel captures the variability in terms of business entities and their relationships, enabling the management of functional variability that involves CRUD operations over the entities, considering one-to-many and one-to-one relationships between them.

Quality Scope. Quality variations are modeled in the `Decision Support` submodel (see Figure 2 top) as quality scenarios where variation points are the stimuli and variants are alternative responses to a stimulus. The `Reference Architecture` submodel (see Figure 2 right bottom corner) is focused on supporting the modeling of architectural

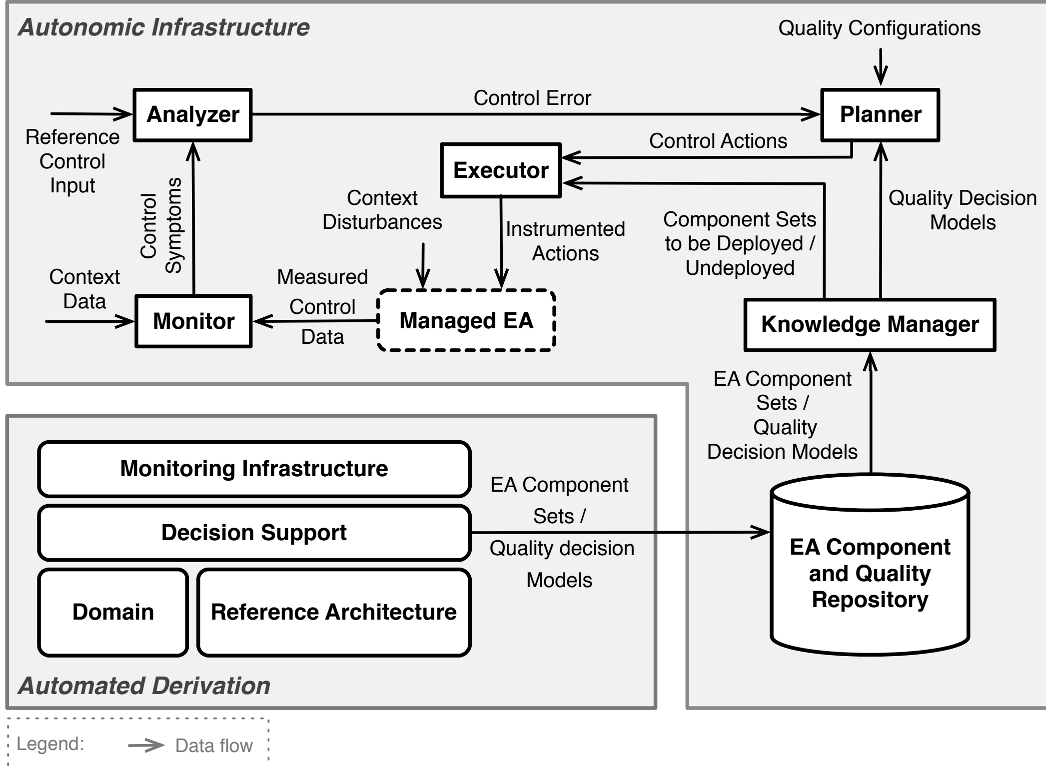


Fig. 1. High-level architectural view of the SHIFT components.

implementations for quality variations. In order to associate architectural implementations for quality variants, we select design patterns in their pure form or we compose them. Resulting structures are documented as variable reference architecture fragments that are later composed and made concrete during the derivation process of components and complete applications. In that way, we compose patterns respecting a base (common) reference architecture, over which variable reference architecture fragments are integrated before deriving concrete implementations. By exploiting the relationships between the `Domain` and `Decision Support` submodels, product line engineers may accurately model the impacts of functional variants on quality attributes, and vice versa, when they exist. In order to manage the suitable relationships between functional and quality variations, from our previous work [14], [15] we consider the need of constraining the bindings between both submodels.

The strategy is based on defining OCL restrictions for capturing and validating constraint logic.

Designing Concrete Architectures. The `Decision Support` submodel provides support for assisted reasoning regarding achievable configurations and their interactions. Our decision model is a collection of (partial) reachable product configurations, expressed as sets of quality variants, and the modeling of their impact on other configurations. The impact of one configuration over another is expressed in terms of *promote*, *require*, *inhibit* and *exclude* relationships. For every pair of related configurations, a reference architecture fragment should be associated. Such fragments model the resulting structures and behavior that produces the composition of patterns associated to variants involved in the related configurations. Concrete architectures of reusable components and complete applications are created as a composition of a common reference architecture and reference architecture fragments. Composition rules are managed in model-based artifacts that will be introduced in the following section.

B. Component Derivation

Reusable components and complete applications result from transforming a set of functionalities contained in a domain model along with a configuration of quality levels into source code. The transformation process satisfies the constraints and conditions dictated by a common reference architecture and the variable reference architecture fragments that contribute. Our generation strategy is based on the delegation of responsibilities for composing templates (*i.e.* model2text transformations) in order to weave common and variable abstract reference architecture fragments. The common reference architecture is associated to a set of controller templates that are in charge of orchestrating the concrete architecture composition. Such controllers know the specific point where a contribution is needed, plus the concrete contribution it requires according to possible variants. Thus, controllers delegate the code declaration to contributors, which are concrete Java classes able

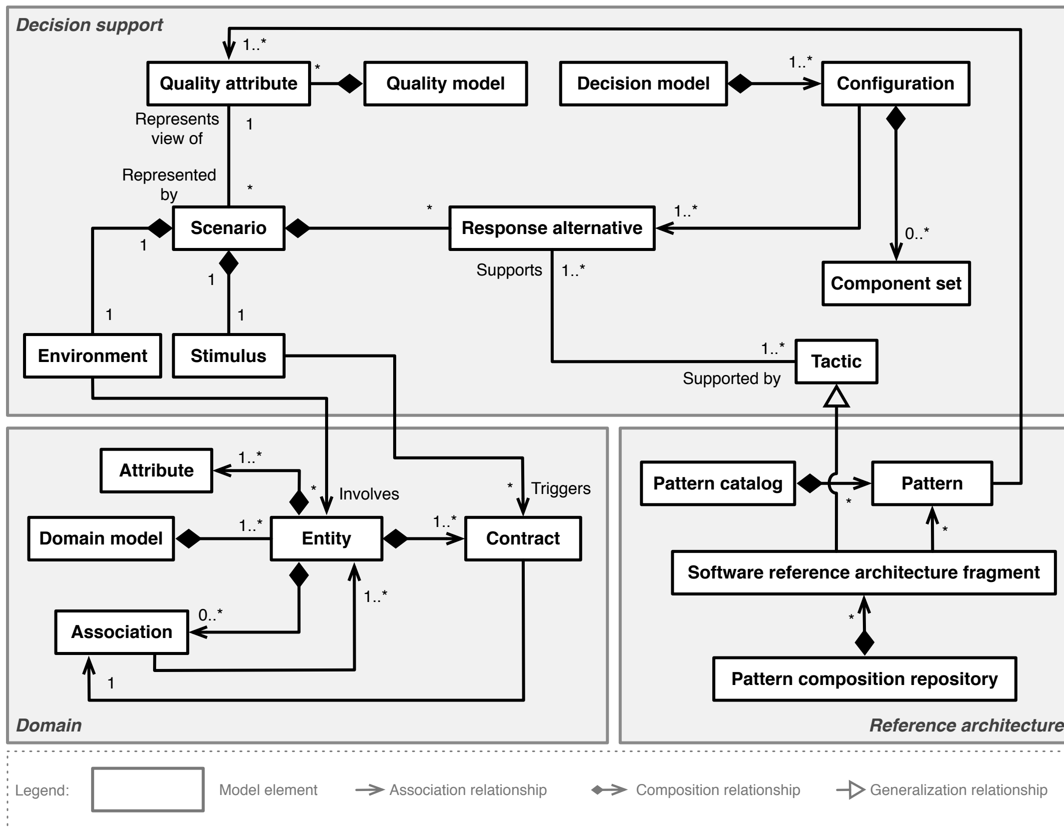


Fig. 2. Model for the derivation of component sets.

to return final source code or delegate on other contributors the responsibility of returning required source code. The generation process also creates templates and contributors for quality variations and related reference architecture fragments. We developed a common library as tool support for describing and weaving required compositions. The library includes a language for describing the required composition, and also includes an engine for weaving code fragments. Currently, we generate JEE7 components under the EJB 3.2 specification. The generation of SCA composites is under development. The specification, design and derivation of quality-concerned enterprise application is part of our recent (unpublished) work available in [16].

C. Specification and Derivation of Monitoring Infrastructure

In SHIFT, the specification and generation of monitoring components, deployable at runtime, is performed through PASCANI. PASCANI is a Domain Specific Language (DSL) we are currently developing that allows defining two types of components: monitor probes (implemented either as an EJB or SCA component) and monitors (implemented as a SCA component). The first are introduced into the system, binding them appropriately and acting as a sensor, therefore allowing to measure actual service executions. Thus, the DSL allows system administrators to monitor components that were not considered to be monitored before the initial system deployment. The second contain the necessary logic to abstract single context events (*i.e.* events arising from monitor probes) into complex and relevant monitoring data to be analyzed by the `Analyzer` and other components (*e.g.*, log components and monitoring dashboards). Both monitor probes and monitors are supplied with standard traceability and controllability mechanisms to (i) prevent the monitoring infrastructure from introducing considerable overhead in the system's regular operations, and (ii) feed knowledge sources with relevant monitoring data. Controlling the produced monitoring components is important when the system reaches critical quality levels, given that it can end up breaching quality agreements or overusing system resources.

The interaction between probes and monitors is event-based, and is specified in a single source file. Monitoring specifications can be parametrized and derived in an automated way for any system component, for those quality attributes with clear definition and already proposed metrics and measurement methods [18]. In SHIFT's current implementation, we have already designed a mechanism for automatically generating PASCANI specifications for the performance quality attribute. This mechanism takes place in the automated derivation phase, and produces the monitoring specification and its corresponding deployment descriptors.

One of most useful features of PASCANI is the standard

abstraction between measurement mechanisms and event-based monitoring logic. This separation of concerns allows PaSCAni to monitor different quality attributes, as monitor probes implementing the necessary measurement methods exist. In our current implementation, monitor probes to measure performance factors are automatically generated and inserted into the system components.

Besides monitor probes and monitors, PaSCAni includes a shared variable model containing relevant monitoring variables holding both reference values (e.g., contracted values in SLAs) and values describing the current state of the system (e.g., current system throughput). Monitors and other components can read and update these values; additionally, they can observe changes in them, by defining events in the monitoring specifications.

In order to monitor EAs, we consider EJB components in our DSL specification, in a way that monitor probes can be integrated with EJB implementations following Aspect Oriented Programming. EJB probes communicate with SCA monitors through Web Service bindings, accomplishing the same functionality that SCA probes. Service interception is realized by means of `Intent` composites, in FraSCAti, and `Interceptors` in the GlassFish application server. Regarding the adaptation of the monitoring infrastructure at runtime, the dynamic deployment is realized by using the dynamic reconfiguration API in FraSCAti, and the *application versioning* feature in GlassFish.

V. Autonomic Infrastructure

The autonomic manager, based on the MAPE-K reference model, is the infrastructure that allows the derived EAs to be adapted to unforeseen context changes in order to ensure the satisfaction of agreed Service Level Agreements (SLA) (see Figure 1). Composing this infrastructure is a `Monitor` element that continuously senses relevant context data, an `Analyzer` that interprets monitoring events reported by the `Monitor` to determine whether the SLAs are being fulfilled, and the `Planner` and `Executor` elements that synthesise and realize adaptation plans to alter the system's behavior, either by modifying the system structure or by varying parameters to reach a desired system state [9]. These four components share relevant information through the `Knowledge` source element.

Our current work considers the automated derivation of the monitoring infrastructure that realizes monitoring elements, comprising (i) monitor probes, attached to the `Managed EA` through a non-intrusive strategy based on aspect oriented programming, and (ii) event-based monitors that collect context data. The `Analyzer` is subscribed to handle monitoring events from `Monitor` elements, and is in charge of deciding when an adaptation is needed to ensure the fulfillment of the performance SLAs.

Our proposed implementation for the `Planner` element follows a constraint satisfaction approach to find the best configuration of components necessary to preserve the fulfillment of the performance SLAs, when available. Finally, the `Executor` element realizes the adaptation plan produced by the `Planner`, redeploying SCA and EJB components by means of the introspection capabilities in the FraSCAti middleware [19] and the *application versioning* feature in the GlassFish application server, respectively.

VI. Automated Reasoning for Components Deployment

The `Planner` element of the `Autonomic Infrastructure` area in Figure 1 includes automated reasoning facilities, which help designing the adaptation plans to alter the system's behavior by modifying its structure or by varying parameters to reach a desired system state. In order to obtain the best possible selection of components to alter the system's behavior, we use the principles of constraint satisfaction to reason on the set of constraints defined by reachable quality scenarios configurations and their relationships with the component sets implementing them.

The following are the definitions we have adapted and established for our reasoning mechanism based on our previous work in [17]. We define a *quality scenarios configuration* as a model consisting of a finite set of `Response Alternatives` with a state of 1, if the response is unselected, or 2, if the response is selected.

We relate on *decision models* the information of components and variants in order to define the necessary actions to derive deployable components in accordance with a quality scenario configuration. Implementing a `Response Alternative` in an application may often require several composed components, thus, we refer as a *component set* to the set of composed components implementing a `Response Alternative`. A decision model is a finite set of decisions, where each decision is a weighted relationship between one *component set* and one `Response Alternative`. A decision may be 0 if the `Response Alternative` does not constraint the deployment of the *component set*, 1 if the *component set* requires the `Response Alternative` to be unselected, and 2 if the *component set* requires the `Response Alternative` to be selected.

A *resolution model* is a decision model instance, which binds variability and defines how to derive one product. Resolution models are the resulting adaptation plans. A resolution model is a finite set of *component set* applications. The application is not planned if the *component set* should not be deployed, and planned if the *component set* should be deployed. However, not every possible resolution model is a valid adaptation plan. A valid adaptation plan must satisfy the following constraints: A *component set* must be deployed satisfying the respective planned application in the decision model. Two deployable *component set*s must not exclude each other. All applicable *component set*s must take into account all the `Response Alternatives`' states in the configuration.

Since many valid adaptation plans may be found, we have formulated [17] some operations on the previous models to provide the `Planner` element with additional information in determining the best possible adaptation plan. The *application* operation takes a *decision model*, a *quality configuration model* and a *resolution model* to verify the resolution model's applicability as an adaptation plan. The *possible resolutions* operation calculates all the potential resolution models from the given quality configuration and decision models. The *number of resolutions* operation calculates the number of potential resolution models from the given quality configuration and decision models. This operation gives an indication of flexibility and complexity of the decision model. The *validation* operation indicates if a given decision model can provide at least one resolution model. The *flexible component sets* operation determines the component sets shared by a given set of possible resolution models. The *inflexible component sets* operation gives the opposite result of the flexible component sets operation, *i.e.* the component sets unique to each resolution model in a set of possible resolution models. The *optimum resolution* operation finds the best resolution model within a set of possible resolution models through the use of a maximizing or minimizing function depending on whether the greater or the least number of component sets, respectively, is more fit to adapt the `Managed EA`.

VII. Conclusions and Future Work

In this paper we have presented our advances on SHIFT, a framework for the generation and management of self-adaptive enterprise applications. We discussed SHIFT's architecture, which is based on the MAPE-K and DYNAMICO reference models [9], [1]. SHIFT copes with two areas: `Automated Derivation` and `Autonomic Infrastructure`. We have illustrated how SHIFT considers changing requirements over SCA composites and EJB models, and offer support for assisted derivation of enterprise applications and monitoring infrastructures. We also discussed how SHIFT offers automated reasoning as part of the `Planner` element in the `Autonomic Infrastructure` area. The `Planner` element is supported on the principles of constraint satisfaction to find the best configuration of components necessary to preserve SLAs. As future work, we will be working on refining the design of the framework and completing the concrete implementations for all the elements presented, including the complete autonomic infrastructure and its interoperability with JEE middlewares.

Acknowledgments

This work has been partially supported by grant 0369-2013 from the Colombian Administrative Department of Science, Technology and Innovation (Colciencias) under project SHIFT 2117-569-33721.

References

[1] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas, "DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems," *LNCS*, vol. 7475, pp. 265–293, 2013.

[2] M. Colombo, E. Di Nitto, and M. Mauri, "Scene: A service composition execution environment supporting dynamic changes disciplined through rules," in *Proc. of the ICSOC'06*. Springer, 2006, pp. 191–202.

[3] L. Baresi and S. Guinea, "Self-supervising bpel processes," *IEEE Trans. on Software Engineering*, vol. 37, no. 2, pp. 247–263, 2011.

[4] N. C. Narendra, K. Ponnalagu, J. Krishnamurthy, and R. Ramkumar, *Run-time adaptation of non-functional properties of composite web services using aspect-oriented programming*. Springer, 2007.

[5] P. Cedillo, J. Gonzalez-Huerta, S. Abrahao, and E. Insfran, "Towards Monitoring Cloud Services Using Models@run.time," in *Proceedings of the 9th Workshop on Models@run.time*, S. Götz, N. Bencomo, and R. France, Eds., Valencia, Spain, 2014, pp. 31–40. [Online]. Available: http://ceur-ws.org/Vol-1270/mrt14_submission_5.pdf

[6] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE Trans. on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2011.

[7] D. Menasce, H. Gomaa, S. Malek, and J. P. Sousa, "Sassy: A framework for self-architecting service-oriented systems," *IEEE Software*, vol. 28, no. 6, pp. 78–85, 2011.

[8] B. Morin, F. Fleurey, N. Bencomo, J.-M. Jézéquel, A. Solberg, V. Dehlen, and G. Blair, "An aspect-oriented and model-driven approach for managing dynamic variability," in *Model driven engineering languages and systems*. Springer, 2008, pp. 782–796.

[9] IBM, "An architectural blueprint for autonomic computing," *IBM White Paper*, 2006.

[10] G. H. Alférez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz, "Dynamic adaptation of service compositions with variability models," *Systems and Software*, vol. 91, no. 1, pp. 24–47, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2013.06.034

[11] R. Heinrich, E. Schmieders, R. Jung, K. Rostami, A. Metzger, W. Hasselbring, R. Reussner, and K. Pohl, "Integrating Run-Time Observations and Design Component Models for Cloud System Analysis," in *Proceedings of the 9th Workshop on Models@run.time*, S. Götz, N. Bencomo, and R. France, Eds., Valencia, Spain, 2014, pp. 41–46. [Online]. Available: http://ceur-ws.org/Vol-1270/mrt14_submission_8.pdf

[12] A. van Hoorn, M. Rohr, A. Gul, and W. Hasselbring, "An adaptation framework enabling resource-efficient operation of software systems," in *Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010*, ser. WUP '09. New York, NY, USA: ACM, 2009, pp. 41–44. [Online]. Available: http://doi.acm.org/10.1145/1527033.1527047

[13] G. Tamura, R. Casallas, A. Cleve, and L. Duchien, "QoS contract preservation through dynamic reconfiguration: A formal semantics approach," *Science of Computer Programming*, vol. 94, pp. 307–332, 2014.

[14] H. Arboleda and J.-C. Royer, *Model-Driven and Software Product Line Engineering*, 1st ed. ISTE-Wiley, 2012. [Online]. Available: http://www.iste.co.uk/index.php?f=x&ACTION=View&id=509

[15] H. Arboleda, R. Casallas, and J.-C. Royer, "Dealing with Fine-Grained Configurations in Model-Driven SPLs," in *Proc. of the SPLC'09*. San Francisco, CA, USA: Carnegie Mellon University, Aug. 2009, pp. 1–10.

[16] D. Durán and H. Arboleda, "Quality-driven software product lines," Master's thesis, Icesi University, 2014.

[17] H. Arboleda, J. F. Díaz, V. Vargas, and J. Royer, "Automated reasoning for derivation of model-driven spls," in *SPLC'10 MAPLE'10*, 2010, pp. 181–188. [Online]. Available: http://splc2010.postech.ac.kr/SPLC2010_second_volume.pdf

[18] ISO/IEC, "ISO/IEC 25000 - Guide to SQuaRE," Tech. Rep., 2014.

[19] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani, "A component-based middleware platform for reconfigurable service-oriented architectures," *Software: Practice and Experience (SPE)*, pp. 1–26, 2012.

View publication stats