Runtime Monitoring and Auditing of Self-Adaptive Systems

Daniel H. Carmo[*], Sergio T. Carvalho[*+], Leonardo G. P. Murta[*], Orlando Loques[*]
[*]Instituto de Computação, Universidade Federal Fluminense (UFF), Niterói, Brazil
[+]Instituto de Informática, Universidade Federal de Goiás (UFG), Goiânia, Brazil
{dheraclio, scarvalho, leomurta, loques}@ic.uff.br

Abstract— **Self-Adaptive Systems are target of frequent research regarding different aspects. However, they still present several challenges related to assurance, dependability, verification, and validation. Adaptations can be related to a set of concerns (i.e., why, what, when, where, who, and how), which are evaluated during, operation and post operation phases. We propose the application of configuration management techniques to provide means for monitoring and auditing Self-Adaptive Systems. We introduce a tool named CM@RT that registers how the system architecture configuration evolves over time and provides different visualizations to track such evolution. For evaluating our approach, some Self-Adaptive Systems scenarios were tackled with the help of CM@RT. The results show that our approach is capable of providing means to perform monitoring and auditing with valuable benefits to the selected Self-Adaptive scenarios.**

Keywords- Self-Adaptive Systems; Configuration Management; Monitoring; Auditing; Product Lines

I. Introduction

Self-Adaptive Systems (SAS) adapt their behaviour in reaction to changes in the runtime context [1]. Today, there is an increasing demand for SAS [2], even for safety-critical applications [3], since SAS are capable of operating in highly dynamic environments [3]. However, considering that SAS are conceived to autonomously react to changes in the runtime context, researches on dependability [4], verification and validation [3], quality [5], among others [1], [2], [6] are crucial, especially for safety-critical applications.

SAS may perform different types of adaptations in response to changes in the runtime context. These adaptation types are based on techniques ranging from parameterization to architecture reconfiguration [2]. The later allows deeper adaptations because parts of the system (i.e., components) can be added, removed, replaced, or reconnected with the remaining parts, resulting in new architecture configurations (AC) [2]. Investigations on AC level adaptation have provided significant results for SAS [7], [8], [9]. For instance, when considering safety-critical applications, the use of Dynamic Software Product Lines (DSPL) techniques leverages dependability by allowing software architects to define, in a preplanned manner, possible AC and transitions among them [10], [11], [12], [13].

However, tracking the AC evolution of SAS, even when adopting DSPL techniques, is a complex task [14]. SAS adaptations can be described in terms of six basic concerns (5W+1H) [2]: *where, when*, *what*, *why*, *who, and how*. Identifying and closely following such concerns is expected during operation phase [2]. In addition to this phase, safety-critical applications also require post-operation analysis, for example, to assign responsibilities in the context of a given health care system failure [6]. In this case, software auditing is essential to verify compliance with design specifications. Therefore, these tasks require specific support for evaluating 5W+1H concerns during two phases:

- *Operation phase*. This could help to answer questions such as "why the system did lots of adaptations in the last week?"
- *Post-operation phase*. This could help answer questions such as "which adaptation patterns led to a system failure?"

A natural way to support such tasks is adopting Configuration Management (CM) techniques during runtime, helping to track the architectural evolution in terms of the 5W+1H concerns. Evolution of AC in SAS and its relation to CM has been explored before. First, van der Hoek [15] made initial efforts to track runtime AC evolution via architecture description languages. After, van der Hoek et al. [16] extended the previous work and applied CM to support DSPL anytime variability. Latter, Georgas et al. [4], [17] leveraged dependability by recording adaptations and providing architecture recovery operations and visualization features over SAS historic behavior. However, such researches do not consider ways for describing adaptations in terms of the main SAS 5W+1H concerns [2], concentrating only on the architectural modifications themselves and ignoring the motivation behind them. This lack of information may jeopardize the comprehension of SAS evolution.

This paper proposes the application of CM techniques to provide support for evaluating 5W+1H concerns with monitoring and auditing purposes. To do so, we designed a CM system to work at runtime, named CM@RT. Our approach comprises two complementary phases: information acquisition and information analysis. First, during information acquisition, CM@RT registers in a repository adaptation related aspects, e.g., runtime contexts and AC achieved. Then, during information analysis, CM@RT supports the visualization of the AC evolution via different perspectives, described in Section III. With this tool, users are able to closely follow the SAS evolution during operation and post-operation.

To evaluate the benefits of CM@RT, some scenarios associated to monitoring and auditing tasks were tackled. With the use of our tool, we were able to collect enough information to answer the 5W+1H concerns for these relevant scenarios.

The remaining sections of this paper are organized as follows. Section II gives more details about SAS and highlights the importance of monitoring and auditing for SAS. Section III introduces the CM@RT approach. Section IV provides some implementation details of CM@RT. Section V presents how to perform monitoring and auditing analyses with CM@RT. Section VI describes some related work. Finally, Section VII presents final considerations and outlines some future work.

II. Self-Adaptive Systems

SAS usually adopt the MAPE-K adaptation loop [2], [5] to manage their behaviour. This adaptation loop consists of four phases [18]: (1) monitoring the external environment in which the system is executing, (2) analysing the context attributes of the environment, (3) planning for a possible adaptation to react from changes in the environment, and (4) enforcing the adaptation in the system. The adaptation loop usually counts on some kind of adaptation knowledge that supports each of the four previously mentioned adaptation phases.

Researchers have explored different techniques to implement adaptation loops. Among them, we can cite model-driven [10], [13], [19] and contract-based (also called strategy or policy) [2], [12], [20] techniques. However, despite of the differences in adaptation techniques, it is possible to describe and further comprehend the adaptation decisions through the evaluation of the 5W+1H concerns [2]:

- *Where*: questions over adaptation location, e.g., which layer or components should be adapted?
- *When:* questions over temporal aspects, e.g., when to adapt or how often?
- *What:* questions over adaptation strategies, e.g., should we reconfigure the AC or substitute a component?
- *Why:* questions over reasons to adapt, e.g., which is the adaptation goal?
- *Who:* questions over sources of adaptation, e.g., was it a human-driven or context-based adaptation?
- *How*: questions over actions performed in the adaptation, e.g., in which order the changes should take place?

Controlling software evolution is one of the main concerns of CM. It traditionally works at development time and has files as first class artifacts [21]. In an usual CM setting, a version control system is responsible for registering detailed information about modifications in different artifacts, e.g., source code [22] and architecture configuration [15]. On the other hand, an issue tracking system is responsible for registering change requests, which identify the issue context, affected artifacts, and required corrections. In addition, integration between version control and issue tracking has demonstrated as an effective way to clarify 5W+1H concerns [23].

The expected operation of a SAS consists on performing adaptations if and only if they are necessary. However, in some situations the SAS may not perform a prescribed adaptation (i.e., false negative) or perform an unnecessary one (i.e., false positive). The evaluation of the internal conditions would help to identify the reasons of false positive (FP) or false negative (FN) adaptations. In addition, replicating the conditions that lead to them is also complex. Appropriate development and test processes would help to identify such problems at development time, before they actually happen. However, it is very difficult to guarantee that all problems are caught during development. This motivates the use of a monitoring mechanism at runtime. Finally, an auditing mechanism could help to identify after-the-fact malfunctions. For example, wrong adaptations (FP or FN), which were not caught at development time or at runtime. This can be useful considering highly dynamic current pervasive computing scenarios.

III. The CM@RT approach

The CM@RT approach performs runtime CM over SAS in order to provide means for evaluating 5W+1H concerns during monitoring and auditing activities. These activities are performed through the integration between the CM@RT-Repository and the CM@RT-Visualizer modules. Technology specific components integrate CM@RT to the SAS infrastructure. Thus, allowing both on-line and off-line analysis. The following subsections present how information is acquired by CM@RT-Repository, and subsequently analyzed through CM@RT-Visualizer.

A. Information Acquisition

The runtime information gathering is performed by the before mentioned integration component. This component is responsible for mapping the application specific SAS entities into the CM@RT-Repository metamodel.

After importing the SAS into CM@RT-Repository, the runtime information captured is registered in the metamodel shown in Figure 1. Its relationship to the 5W+1H concerns is described in the following.

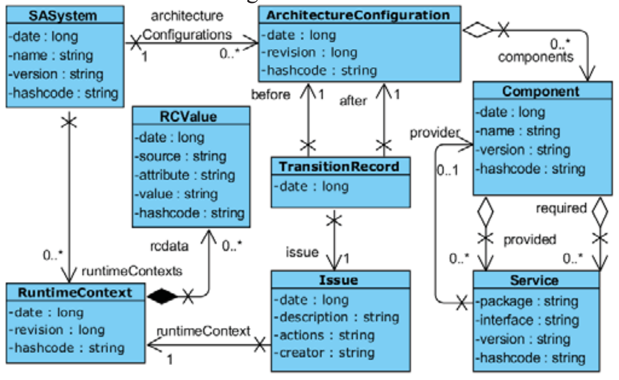


Figure 1 CM@RT metamodel

To answer *where the adaptation should be performed*, CM@RT tracks *Architecture Configurations* materialized during runtime. This tracking enables comparing AC to reveal the differences between them. The relationship between the AC before and after adaptation is represented by the *Transition Record* entity. It also includes references to the *Issue* entity, which is described further in following paragraphs.

The answer to *when to adapt* depends on when the demand arose and when it was detected. Since SAS are context-aware, the demand appears first on the runtime context, represented by the *Runtime Context* entity. The moment of the demand detection is the time of the *Issue* entity creation. However, not every runtime context results in new demands. In addition, *when to adapt* involves deciding whether to adapt or not, so demands may go unattended. The CM@RT-Repository stores *Runtime Contexts* and *Issues* even if they do not result in AC transitions.

The answer to *who detected the issue* can be found in the attribute *creator* of the *Issue* entity, and it may consist on the SAS itself or on a human. SAS issues are generally concerned with self-* properties [21], e.g., self-healing, or user needs, e.g., controlling room temperature. On the other hand, human-driven issues may have different concerns, e.g., corrective maintenance during operation, testing components during development, or evaluating system behaviour after adaptations. In any case, they relate to the current runtime context and determine modifications in the AC.

The answer to *why the* SAS *adapts* requires evaluation over the *Runtime Context* and the *Issue* associated to a transition between AC. The runtime context shows detailed information over SAS runtime environment, allowing engineers to reason over them. Issues deal with several concerns (discussed in the previous paragraph), which can be registered in the attribute *description* for future analysis. Thus, *Runtime Contexts* and the corresponding *Issues* complement each other for answering this question.

The answer to *how the adaptation occurred* is also associated with the *Issue* through adaptive actions it registers. However, these actions may consist on high-level modifications over the current AC, which leave aside low-level dependencies among components and required services. For example, contracts are concerned with DSPL composition rules only [24], thus not required to address version conflicts of shared libraries among components. It is necessary to evaluate the differences between AC before and after the adaptations to realize the full extent of *how* the SAS was adapted. This could reveal if the SAS low-level dependency solution harmed contract results.

B. Information Analysis

The CM@RT-Visualizer is intended to work connected to CM@RT-Repository, retrieving information from its repository and exposing update events. Through a combination of views, the application provides analysis features that can be applied during *monitoring* and *auditing*.

Monitoring support allows software architects and engineers to evaluate system behavior during runtime. For example, they may evaluate transitions among AC online, since the CM@RT-Visualizer updates its representations just after adaptations are recorded in CM@RT-Repository. It is also possible to evaluate the runtime contexts and issues as soon as they are recorded.

Auditing support is only viable because all information is recorded in a repository, allowing after-the-fact analysis. It includes, among other features, a retrospective view. It is able to graphically replay the transitions performed at runtime in terms of AC diffs, runtime contexts diffs, and associated issues (see Section IV.D for more details).

IV. CM@RT PROTOTYPE

With the aim of evaluating our approach, we develop four prototypes. The first is a SAS framework, which follows the MAPE-K principle. The second is a version of the SCIADS[1] [12], [24], [26] compliant to the SAS framework. The third is the CM@RT-Repository module and the fourth is the CM@RT-Visualizer module.

The main adaptation technology employed in our prototypes is OSGi[2]. OSGi leverages the construction of dynamic systems as sets of components, which can bind to services dynamically and automatically. The following subsections describe the developed prototypes.

A. The SAS Framework

The SAS Framework was implemented as independent OSGi components. The main component is the SelfAdapter. It supplies the basis of the framework, defining service interfaces, and model classes, and providing utility classes for others SAS Framework components. Other OSGi components (Knowledge, Monitor, Analyzer, Planner, and Adapter) implement specific services following the interfaces required by the SelfAdapter component. For example, the Monitor component implements the monitoring service of MAPE-K defined by the Monitor interface.

B. The SCIADS Version Compliant to the SAS Framework

SCIADS is a safety-critical DSPL-based SAS. Its AC is reconfigured according to adaptation actions determined by contracts and DSPL composition rules [12], [24]. The contracts use runtime context variables to determine high-level modifications in current AC, e.g., insertion or removal of components. In addition, SCIADS AC considers its patient specifics needs and residence characteristics. The combination of these factors results in a great number of possible AC and directly jeopardizes the predictability of system states [13].

C. The CM@RT-Repository Module

The CM@RT-Repository information acquisition module is the main element of our approach, as it provides services to all the other elements. The CM@RT-Repository provides a single API, which follows the Facade pattern, hiding internal services and their relationship.

There are two service groups: repository and diffing. Repository services include SAS registration, information

[1] SCIADS is a home health care system developed at UFF (http://www.tempo.uff.br/sciads/).

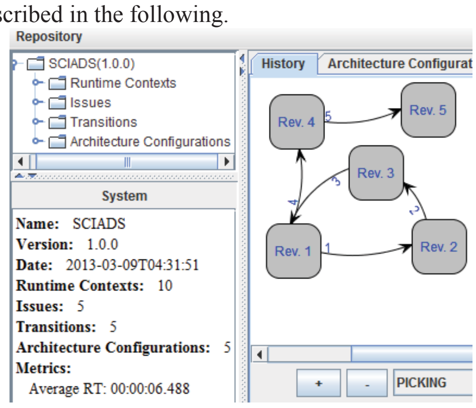
[2] Official site: http://www.osgi.org

storage, and querying over metamodel elements such as runtime context, issues, AC, and theirs transitions (see Figure 1). The diffing service supports runtime context and AC comparison.

D. The CM@RT-Visualizer Module

The CM@RT-Visualizer module depends on services provided by the CM@RT-Repository module. For example, the query services of CM@RT-Repository are used to populate the views of CM@RT-Visualizer.

After selecting a SAS among those registered in the repository, the information available for analysis is shown in the **Repository view**. It is organized in four groups: Runtime Contexts, Issues, Transitions, and Architecture Configurations. These groups can be seen in Figure 2, on the left hand side. Selected or general information can be analyzed through seven different views during monitoring or auditing activities, which are described in the following.



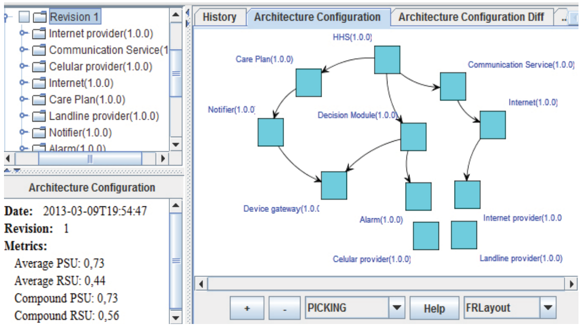
Name: SCIADS
Version: 1.0.0
Date: 2013-03-09T04:31:51
Runtime Contexts: 10
Issues: 5
Transitions: 5
Architecture Configurations: 5
Metrics:
Average RT: 00:00:06.488

Figure 2 History view

The **History view** presents the SAS evolution as a whole through a graph. Figure 2 shows an example of runtime evolution history, in which SCIADS performed four different transitions. The nodes represent AC and the edges represent the transitions.

The **Architecture Configuration view** presents the SAS AC also through a graph notation (see Figure 3). In this case, the nodes represent components and are identified by name and version, while edges represent the AC topology. Figure 3 also shows metrics for assessing the quality of the AC [27]. The metrics are based in component provided service utilization (PSU) and required service utilization (RSU). The metrics are Average PSU, Average RSU, Compound PSU, and Compound RSU.

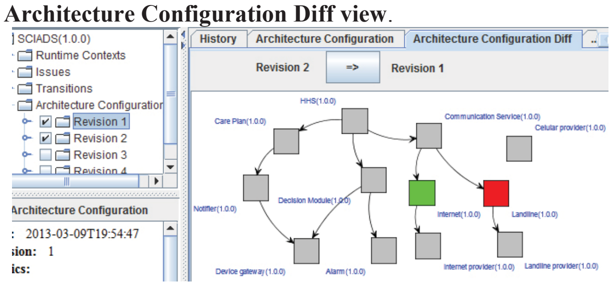
The **Architecture Configuration Diff view** (see Figure 4) shows the differences between two AC selected from the **Repository view** through check boxes. It uses the same graph representation used in **Architecture Configuration view**, annotated with color codes, indicating added (green), removed (red), replaced (blue), and unchanged components (gray). In addition, there is a textual description identifying the differences found between the selected AC. This feature also uses the before mentioned color codes to favor visual identification.



Date: 2013-03-09T19:54:47
Revision: 1
Metrics:
Average PSU: 0,73
Average RSU: 0,44
Compound PSU: 0,73
Compound RSU: 0,56

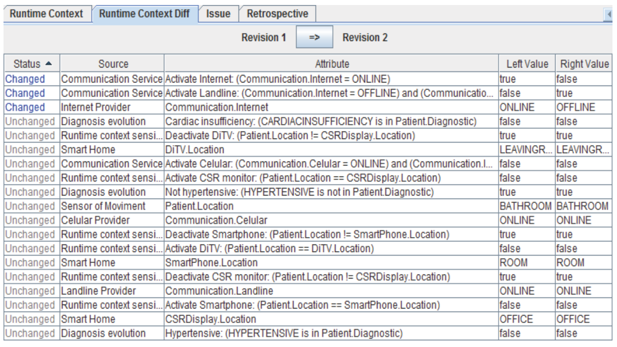
Figure 3 Architecture Configuration view

The **Runtime Context view** represents the context information monitored by the SAS in the form of a grid. This grid contains the date of last update, the source, the attribute name, and the collected value. Similarly, the **Runtime Context Diff view** (see Figure 5) shows the differences between two runtime contexts selected from the Repository view. The diffing is performed considering each attribute and its value, but ignoring dates and sources, as these data are expected to always change. The Status column shows the results using the same designation and color code of the **Architecture Configuration Diff view**.



: 2013-03-09T19:54:47
sion: 1
ics:

Figure 4 Architecture Configuration Diff view

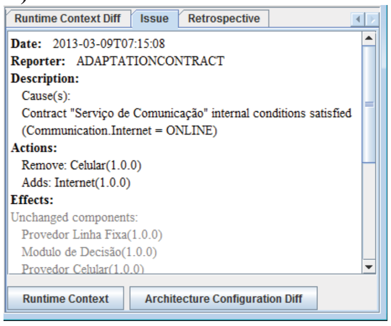


Status	Source	Attribute	Left Value	Right Value
Changed	Communication Service	Activate Internet: (Communication.Internet = ONLINE)	true	false
Changed	Communication Service	Activate Landline: (Communication.Internet = OFFLINE) and (Communicatio...	false	true
Changed	Internet Provider	Communication.Internet	ONLINE	OFFLINE
Unchanged	Diagnosis evolution	Cardiac insufficiency: (CARDIACINSUFFICIENCY is in Patient.Diagnostic)	false	false
Unchanged	Runtime context sensi...	Deactivate DiTV: (Patient.Location != CSRDisplay.Location)	true	true
Unchanged	Smart Home	DiTV.Location	LEAVINGR...	LEAVINGR...
Unchanged	Communication Service	Activate Celular: (Communication.Celular = ONLINE) and (Communication.I...	false	false
Unchanged	Runtime context sensi...	Activate CSR monitor: (Patient.Location == CSRDisplay.Location)	false	false
Unchanged	Diagnosis evolution	Not hypertensive: (HYPERTENSIVE is not in Patient.Diagnostic)	true	true
Unchanged	Sensor of Moviment	Patient.Location	BATHROOM	BATHROOM
Unchanged	Celular Provider	Communication.Celular	ONLINE	ONLINE
Unchanged	Runtime context sensi...	Deactivate Smartphone: (Patient.Location != SmartPhone.Location)	true	true
Unchanged	Runtime context sensi...	Activate DiTV: (Patient.Location == DiTV.Location)	false	false
Unchanged	Smart Home	SmartPhone.Location	ROOM	ROOM
Unchanged	Runtime context sensi...	Deactivate CSR monitor: (Patient.Location != CSRDisplay.Location)	true	true
Unchanged	Landline Provider	Communication.Landline	ONLINE	ONLINE
Unchanged	Runtime context sensi...	Activate Smartphone: (Patient.Location == SmartPhone.Location)	false	false
Unchanged	Smart Home	CSRDisplay.Location	OFFICE	OFFICE
Unchanged	Diagnosis evolution	Hypertensive: (HYPERTENSIVE is in Patient.Diagnostic)	false	false

Figure 5 Runtime Context Diff view

The **Issue view** shows details of the registered issues. Figure 6 shows the inclusion date in the repository, the demand description, the actions required to modify the AC, and a textual description of the effects over the AC before the adaptation. It also has buttons to show the runtime context (**Runtime Context view**) and the difference between the AC before and after the adaptation (**Architecture Configuration Diff view**).

Finally, the **Retrospective view** uses the transition records from CM@RT-Repository to present an animation of the adaptations performed by the SAS system, replaying its operation for a chosen period. It is a composition of three views: **Architecture Configuration Diff view** (see Figure 4), **Runtime Context Diff view** (see Figure 5), and **Issue view (see Figure 6)**.



Date: 2013-03-09T07:15:08
Reporter: ADAPTATIONCONTRACT
Description:
Cause(s):
Contract "Serviço de Comunicação" internal conditions satisfied (Communication.Internet = ONLINE)
Actions:
Remove: Celular(1.0.0)
Adds: Internet(1.0.0)
Effects:
Unchanged components:
Provedor Linha Fixa(1.0.0)
Modulo de Decisão(1.0.0)
Provedor Celular(1.0.0)

Figure 6 Issue view

V. MONITORING AND AUDITING ANALYSES WITH CM@RT

This section describes how CM@RT can help performing monitoring and auditing analyses under three scenarios. These scenarios use SCIADS as a concrete example. In this section, we assume that all prototypes are active during runtime in the OSGi platform. Monitoring and auditing performed using CM@RT-Visualizer features occurs as follows.

A. False Positive and False Negative Adaptations Detection

In SCIADS, identifying FP or FN adaptations require the evaluation of several contracts, considering their internal conditions. In Figure 5, the CM@RT-Visualizer shows examples of these internal conditions in the first 2 rows, along with corresponding values. The internal conditions were added to runtime context to enable their evaluation based in value differences between two runtime contexts.

To identify a FP the user can also check the registered issues. The Issue view reveals which demands the SAS found on a particular runtime context in the description field. To identify FN, it is necessary to go through the registered runtime contexts. The Runtime Context Diff view helps to identify significant changes between runtime contexts, necessary to locate situations where adaptations should have occurred but did not.

B. Adaptation Cycles Detection

Another possible monitoring scenario consists on evaluating the SCIADS adaptation rate in a patient's home. Besides adopting massive tests at development time, each patient home has its own requirements and features. Sometimes these features cannot be fully predicted, thus being a source of uncertainty to the SAS [28]

Monitoring new AC transitions thought CM@RT-Visualizer reveals the adaptation rate. Furthermore, it is possible to monitor the rate of runtime context updates and issues found. Monitoring runtime context and issues could reveal anomalies during operation which were missed during development. For example, the temperature thresholds configured according to the development site may be inadequate to the operation site, leading patient discomfort.

C. After-the-fact Adaptations Tracking

In SCIADS, if the patient is under dangerous health conditions, adaptations are severely restricted. During auditing of such behaviour, the manual analysis of textual system runtime logs would be counterproductive and error prone. The existence of a tool capable of representing historical information with semantic driven visualizations, make the auditing process more efficient and trustable if compared to textual analysis.

The Retrospective view represents the progression of the SAS adaptations, enabling their evaluation by specialists. The view shows at the same time what changed, why it did, when it happened, and who requested it. In addition, the progression of the runtime context may reveal the effect of the adaptations in the SAS environment.

VI. RELATED WORK

Few works provide runtime CM infrastructure to manage AC evolution. Van der Hoek et al. [15], show that the AC of dynamic system evolve as well as source code, and were the first to propose the use of CM for managing this evolution. They developed an integrated architecture-driven environment called Mae. Mae provides features such as architecture evolution control, runtime adaptation patch generation, and product variant selection. In [16], van der Hoek continues to explore architecture evolution control for any time variability on DSPL. The approach comprises two applications: Ménage for evolution control and SelectorDriver for evolution handling. Ménage is part of the Mae environment, but focus only on the development phase. Our approach is complementary to these in the sense that we propose an infrastructure for tracking the architectural evolution during operation and post-operation phases, while their approach focuses on the development phase.

Georgas et al. [4] use runtime CM to control architectural evolution and to leverage dependability on SAS through the use of Architectural Runtime Configuration Management (ARCM). ARCM is integrated to Eclipse IDE through a plugin [4], and comes with three main features: runtime architectural evolution control, graph-based visualization of architectural evolution, and architectural recovery facilities. In [17], Georgas et al. describe the use of ARCM to provide visibility and understandability over SAS runtime behavior and means for human intervention over the adaptation process. Despite the fact that it has some similarities with our approach, it is limited to AC evolution. Our approach tracks several other architectural evolution concerns, such as runtime context and related issue, and encompasses operation and post-operation phases.

VII. CONCLUSION

CM@RT represents our initial efforts on supplying CM at Runtime to provide a monitoring and auditing infrastructure for SAS, with special attention to DSPL. CM@RT-Repository provides tracking functionalities over AC evolution and related information. Complementing the core application, we provide a repository visualization tool that supplies consolidated information and mechanisms for monitoring and auditing. Thus, CM@RT enables short, medium, and long time analysis over SAS behavior. In addition, since the CM@RT was designed to be deployed with the target SAS, it runs on-line in production and development environment.

We also demonstrated how to perform monitoring and auditing with CM@RT. This provides some initial evidences that CM@RT is capable of providing behavior information for monitoring and trace information for auditing. In addition, we provide some SPL metrics for quality assessment and analysis support (see Figure 3). In the future, we intend to perform user-centered experiments to raise more evidence of its benefits.

We believe the application of data mining on the repository would reveal the existence of significant adaptation patterns. For example, policies conflicts, components interoperability conflicts, or singular situations at runtime environment could be detected. With these patterns in hand, it would be possible to enhance the effectiveness of the adaptation contracts by treating previous patterns causes and, consequently, avoiding unstable architecture configurations. In addition, there are plans to use this information for developing automated analysis features.

Acknowledgment

The authors would like to thank CAPES, CNPq, and FAPERJ for their financial support.

References

[1] B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, and others, "Software engineering for self-adaptive systems: A research roadmap," *Software Engineering for Self-Adaptive Systems*, pp. 1–26, 2009.

[2] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 1–42, May 2009.

[3] G. Tamura, N. M. Villegas, H. A. Müller, J. P. Sousa, B. Becker, M. Pezzè, G. Karsai, S. Mankovskii, W. Schäfer, L. Tahvildari, and Wong, Kenny, "Towards practical runtime verification and validation of self-adaptive software systems," in in *Software Engineering for Self-Adaptive Systems 2*, vol. 7475, Springer, 2013, pp. 108–132.

[4] J. C. Georgas, A. Van Der Hoek, and R. N. Taylor, "Architectural runtime configuration management in support of dependable self-adaptive software," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–6, Maio 2005.

[5] N. M. Villegas, H. A. Müller, G. Tamura, L. Duchien, and R. Casallas, "A framework for evaluating quality-driven self-adaptive software systems," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Waikiki, Honolulu, HI, USA, 2011, pp. 80–89.

[6] O. Loques and A. Sztajnberg, "Adaptation issues in software architectures of remote health care systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care*, 2010, pp. 24–28.

[7] D. Garlan, B. Schmerl, and S. W. Cheng, "Software architecture-based self-adaptation," *In Autonomic computing and networking*, no. Springer, pp. 31–55, 2009.

[8] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimhigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *Intelligent Systems and Their Applications, IEEE*, vol. 14, no. 3, pp. 54–62, 1999.

[9] B. Hayes-Roth, "An architecture for adaptive intelligent systems," *Artificial Intelligence: Special Issue on Agents and Interactivity*, vol. 72, pp. 329–365, 1995.

[10] N. Bencomo, P. Sawyer, G. Blair, and P. Grace, "Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems," in *2nd International Workshop on Dynamic Software Product Lines*, Limerick, Ireland, 2008, vol. 2, pp. 23–32.

[11] T. Dinkelaker, R. Mitschke, K. Fetzer, and M. Mezini, "A Dynamic Software Product Line Approach Using Aspect Models at Runtime," *5th Domain-Specific Aspect Languages Workshop*, Mar. 2010.

[12] S. T. Carvalho, O. Loques, and L. Murta, "Dynamic Variability Management in Product Lines: An Approach Based on Architectural Contracts," presented at the IV Brazilian Symposium on Software Components, Architectures and Reuse, Bahia, Brazil, 2010, pp. 61–69.

[13] B. Morin, O. Barais, J.-M. Jézéquel, F. Fleurey, and A. Solberg, "Models@ Run.time to Support Dynamic Adaptation," *IEEE Computer*, vol. 42, no. 10, pp. 44–51, Oct-2009.

[14] N. López, R. Casallas, and A. Van Der Hoek, "Issues in mapping change-based product line architectures to configuration management systems," in *Proceedings of the 13th International Software Product Line Conference*, 2009, pp. 21–30.

[15] A. Van Der Hoek, M. Mikic-Rakic, R. Roshandel, and N. Medvidovic, "Taming Architectural Evolution," in *ACM SIGSOFT Software Engineering Notes*, 2001, vol. 26, pp. 1–10.

[16] A. Van Der Hoek, "Design-Time Product Line Architectures for Any-Time Variability," *Science of Computer Programming*, vol. 53, no. 3, pp. 285–304, 2004.

[17] J. C. Georgas, A. Van Der Hoek, and R. N. Taylor, "Using Architectural Models to Manage and Visualize Runtime Adaptation," *IEEE Computer*, vol. 42, no. 10, pp. 52–60, 2009.

[18] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[19] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *IEEE Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[20] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *IEEE Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[21] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," International Business Machines Corporation, 278606109, 2001.

[22] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version Control with Subversion*, 2nd ed. Sebastpol, CA, USA: O'Reilly Media, 2008.

[23] C. R. Dantas, L. G. P. Murta, and C. M. L. Werner, "Mining Change Traces from Versioned UML Repositories," in *Brazilian Symposium on Software Engineering (SBES)*, João Pessoa, Brazil, 2007, pp. 236–252.

[24] S. T. Carvalho, L. Murta, and O. Loques, "Variabilities as First-Class Elements in Product Line Architectures of Homecare Systems," in *4th International Workshop on Software Engineering in Health Care*, Zurich, Switzerland, 2012, pp. 33–39.

[25] K. Suzaki, T. Yagi, K. Iijima, N. A. Quynh, C. Artho, and Y. Watanebe, "Moving from logical sharing of guest OS to physical sharing of deduplication on virtual machine," in *Proc. 5th USENIX Workshop on Hot Topics in Security (HotSec 2010), USENIX, Washington DC, USA*, 2010.

[26] S. T. Carvalho, A. Copetti, and O. Loques, "Ubiquitous Computing System in Home Health Care," *Journal of Health Informatics*, vol. 3, no. 2, pp. 51–57, 2011.

[27] A. Van Der Hoek, E. Dincel, and N. Medvidovic, "Using service utilization metrics to assess the structure of product line architectures," in *Software Metrics Symposium, 2003. Proceedings. Ninth International*, 2003, pp. 298–308.

[28] N. Esfahani and S. Malek, "Uncertainty in Self-Adaptive Software Systems," in in *Software Engineering for Self-Adaptive Systems 2*, vol. 7475, Springer, 2012, pp. 214–238.