Towards Autonomic Context-Aware Computing for SaaS Through Variability Management Mechanisms

Asmae Benali, Bouchra El Asri and Houda Kriouile

IMS Team, SIME Laboratory
ENSIAS, Mohammed V University
Rabat, Morocco
{asmae.benali, houda.kriouile}@um5s.net.ma
elasri@ensias.ma

***Abstract*—Owing to the multi-tenancy of Software-as-a-Service applications, the management of their resources becomes a challenge and a crucial task in order to provide highly configurable applications to thousands of tenants in a shared and heterogeneous cloud environment. They need dynamic context-aware configuration and intelligent strategies for provisioning available and cost-efficient services. In this sense, this paper identifies open issues in autonomic resource provisioning and shows innovative management techniques for these applications on cloud. Indeed, our work will focus on implementing an autonomic management artifact of services variability concerning the context. In this paper, we highlight our process for the development of autonomic context-aware to manage the SaaS variability.**

Keywords--multi-tenancy; context-aware; autonomic system; SPL; SaaS

I. INTRODUCTION

The emergence of SaaS (Software-as-a-Service) provision and cloud computing in general had recently a tremendous impact on corporate information technology.

While the implementation and successful operation of powerful information systems continues to be a corner stone of success in modern enterprises, the ability to acquire IT (Information Technology) infrastructure, software, or platforms on a pay-as-you-go basis has opened a new avenue for optimizing operational costs and processes. Cloud computing as defined by the NIST [1] as an IT model that allows network to have an easy access to a shared set of configurable computing resources. Cloud Computing providers offer their services in three basic models: SaaS, PaaS (Platform-as-a-Service) and IaaS (Infrastructure-as-a-Service).

A SaaS application is hosted by a provider in the cloud, rented to multiple tenants and accessed by the tenants' users over the Internet [2]. Also, application resources are shared among tenants. In the provisioning of a SaaS application, various stakeholders with different objectives are involved, i.e., providers of all cloud stack layers as well as tenants and their users [1].

Hence, an autonomic and dynamic configuration management is necessary in order to offer these highly configurable SaaS applications.

Some configuration steps, e.g., performed by tenants, are independent from each other. However, others are dependent, e.g., tenant's configuration choices depend on the pre-configuration of the provider. Thus, these later depend on the context-aware of the providers.

In addition, stakeholders' objectives may change over time, e.g., if a tenant decides to change the tenancy contract. Thus, the configuration process needs to support reconfiguration of stakeholder pre-configurations and subsequent ones being further affected.

Our ongoing works are twofold. Firstly, we define context-aware for a configuration management of SaaS applications. Secondly, we suggest an autonomic configuration management based on SPLE (Software Product Line Engineering) [3].

The structure of this paper is as follows. We describe the background in Sections II and III. Then, we show our motivations in Section IV. Section V depicts our futures contributions. In Section VI, we present the related work and the state-of-art. Finally, we conclude this paper in Section VII.

II. VARIABILITY-AWARE SYSTEM

Variability is an ability of software artifacts that allows them to be extended, modified, customized or configured to meet specific needs [4]. In this section, we discuss, in general, the literature concerning systems based on variable modules. Several works have been proposed. We have classified them according to the different phases of software engineering, namely, elicitation time, design time, compile time and binding time. The system variability may occur in all these phases [5].

A. Elicitation Time

It is precisely about managing the variability at the customer's requirements level, examining their priorities and making appropriate choices. A variety of requirement approaches have been proposed in recent works. Barney et al. [6] showed that the management of software product value depends on the context in which the product exists.

B. Design Time

At design time, all variants and variations points are defined in the software architecture or in a complementary feature tree or table. Several approaches were proposed in this phase to model software product lines by using feature models starting with the FODA (Feature Oriented Domain

Analysis) approach [7]. This approach aims at capturing the commonalities and differences points at requirement level.

C. *Compilation Time*

During the compilation time, the variability described in the architecture must be compiled in the software components (e.g., core assets in a product line) by means of a variety of programming techniques. Cardelli et al. [8] proposed a framework where each module is separately compiled to a self-contained entity and showed that this separation makes it possible to link safely the compatible modules together.

D. *Binding Time*

Binding time is a property of variation points to delay the design decisions to a later stage, as new requirements or different context conditions may require concretize the variability at any time after design time. Trummer [9] introduced a corresponding data model that is based upon the Café (Cloud Application Framework) model. Applications are composed out of components that may be provisioned separately.

III. CONTEXT-AWARE SYSTEM

An understanding of how context can be used will help us determine what context-aware behaviors to support in our future framework [5].

A. *Context*

Before specifying our own definition of context to use, we will look at how researchers have defined context in their own work. The first work that introduced the term ‘context-aware’ was done by Schilit and Theimer [10]. They defined context as location, identities of nearby people and objects, and changes to those objects. Dey et al. [11] defined context as:”*... any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*.”

In our work, we will adopt this definition because it allows context to be either explicitly or implicitly indicated by the user.

B. *Context-Aware System*

The first research investigation of context-aware computing was discussed by Want et al. [12] in 1992. Since then, numerous approaches attempts to define context-aware computing were appeared. Hull et al. [13] defined context-aware computing to be the ability of computing devices to detect and sense, interpret and respond to aspects of a user's local environment and the computing devices themselves. Dey and Abowd [14] defined Context-Aware as:”A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task”. In our work, we will adopt this definition because it remains the most generic.

IV. MOTIVATIONS: THE NEED OF AUTONOMIC COMPUTING FOR THE SAAS ACCORDING TO THE TENANT-CONTEXT

SPL have become a common skill for creating software systems that share a common set of commonalities and variabilities that distinguish specific products, thus promoting the development of a family of related products.

Deploying an application in the cloud provides to its owner many advantages: cost reduction, scalability, high availability, etc. However, the migration of an application or the development of a new service in the cloud is not trivial because of the large number of functional and non-functional requirements to deal with [5].

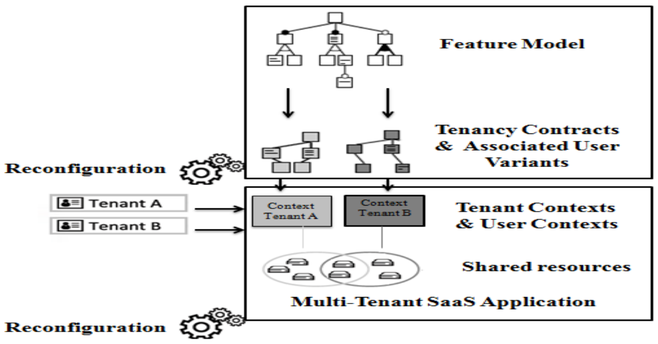

Figure 1. Configuration and instansiation of SaaS application.

We show in Figure 1 how a multi-tenant SaaS application is configured. Tenancy contracts define the provisioned application functionality as well as QoS (Quality of Service) guarantees. Thus, an Extended domain Feature Model (EFM) [15] with attributes is convenient to express this variability and a staged configuration as proposed by Czarnecki et al. is applicable to create those contracts [16]. In contrast to conventional SPL engineering, multiple tenancy contracts and user variants are derived, but integrated into a single application instance in the solution space. To handle this variability, a self-adaptive application architecture was proposed. In this paper, we focus on autonomic managing the variability of SaaS applications by taking into account the context-aware of the system.

V. TOWARD AUTONOMIC CONTEXT-AWRE MANAGEMENT OF VARIABILY

In this section, we will present the notion of autonomic system, and our overview process to achieve autonomic configuration.

A. *Autonomic Systems*

Autonomic systems are self-regulating, self-healing, self-protecting, and self-improving [17]. Therefore, Autonomic computing capabilities can address the adaptation and reconfiguration challenges of the SaaS cloud layer. Some key open challenges are:

- Self-configuring: As stakeholder objectives change, e.g., if a tenant decides to rent different functionality, the tenant’s configuration needs to be reconfigured.

- QoS: Cloud Service Providers (CSPs) need to ensure that sufficient amount of resources is provisioned to ensure that QoS requirements of CSCs (Cloud Service Consumers), such as deadline, response time, and budget constraints are met.
- Security: Achieving security features such as availability. If a coordinated attack is launched against the SaaS provider, the sudden increase in traffic might be wrongly assumed to be legitimate requests and resources would be scaled up to handle them.

B. Overview of our Process

Our autonomic system of management variability is presented in Figure 2.

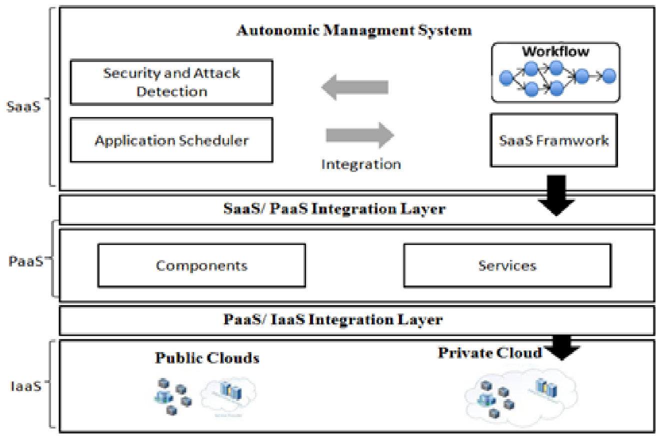
Autonomic Managment System
Security and Attack Detection
Workflow
Application Scheduler
Integration
SaaS Framwork
SaaS
SaaS/ PaaS Integration Layer
PaaS
Components
Services
PaaS/ IaaS Integration Layer
Public Clouds
Private Cloud
IaaS

Figure 2. System architecture for autonomic cloud management.

- Application Scheduler: The scheduler is responsible for assigning each task in an application to resources for execution based on user QoS parameters and the overall cost for the service provider.
- Security and Attack Detection: This component implements all the checks to be performed when requests are received .

The workflow of the process proposed which is depicted in Figure 3.

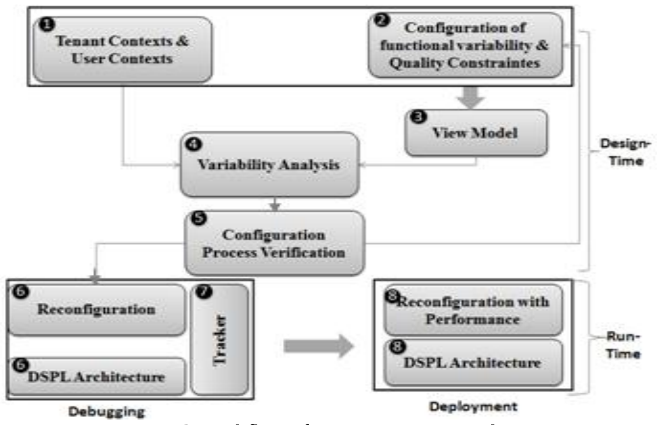
1 Tenant Contexts & User Contexts
2 Configuration of functional variability & Quality Constraintes
3 View Model
4 Variability Analysis
5 Configuration Process Verification
Design-Time
6 Reconfiguration
6 DSPL Architecture
7 Tracker
Debugging
8 Reconfiguration with Performance
8 DSPL Architecture
Deployment
Run-Time

Figure 3.Workflow of our process proposed.

- Step 1: Specifies the context of the reconfigurable system.

User variant configurations are instantiated as user contexts in the SaaS application instance. The users of a tenant have their own user context, each conforming to a user variant configuration. The context of the reconfigurable systems is specified by means of the OWL (Web Ontology Language) [18]. This language provides a vocabulary for describing system context knowledge and for specifying conditions in the context.

- Step 2: Specifies the variability and commonality among functionality and quality properties

The stakeholders have varying requirements on functionality and QoS. Therefore, we need to handle the variability of both. Stakeholders' objectives consider functional variability and variability among quality constraints, e.g., performance, availability, and the server location. We will use an EFM with mixed constraints and group cardinalities.

- Step 3: defines stakeholders and their views on the extended feature model

A stakeholder either represents a person, a member of an organization, or a third party that is involved in the configuration process and has certain concerns regarding the configuration of parts of the EFM. Views are defined by mapping configuration operations specified for the EFM onto groups and categories specified in the View Model. This later defines stakeholders and their views on the extended feature model [19].

- Step 4: Analyzes the reconfigurations before performing them.

Process verification needs to ensure that the configuration process is consistent with the EFM. This is needed for error-correction and avoidance while it would also help users keeping track of their configurations.

- Step 5: Analysis results.

After the given analysis results, the previous configuration can be updated or leveraged at run-time phase.

- Step 6: To Debugs the run-time reconfigurations.

Given the fact that not all potential run-time failures can be anticipated during system design, it is possible to set up MoRE (Model-based Reconfiguration Engine) [20] with a debugging-enabled reconfiguration strategy. This strategy keeps the history of system configurations.

- Step 7: Keeps track of the reconfigurations.

In the context of experimentation, MoRE can store trace entries about the reconfigurations. This provides information for a posterior analysis, which ranges from context conditions to reconfiguration plans.

- Step 8: To deploy the system in the target platform.

Once the development is finished, there is no interest in debugging information any longer. Therefore, MoRE can be set up with another reconfiguration strategy which lacks debugging support but achieves better performance. We suggest using MoRE featuring a performance-oriented reconfiguration strategy tool.

VI. RELATED WORK

This section presents work that is related to the concepts of our configuration management, which copes with different research fields. Mietzner et al. propose using SPL techniques for configuring multi-tenant SaaS applications [21]. The tenant's configuration decisions are influenced by already deployed services. Concerning our approach, tenants' pre-configurations are not influenced by the configuration of new tenants. Cheng et al. [22] apply SPL techniques on configurable SaaS applications. The description of the application flexibility is created in domain engineering. This catalog is then used to configure the application per tenant. In contrast, we will use EFMs to model the functionality of the application as well as QoS and assume the context-aware of the tenant. Another concept which describes variability for SaaS applications is given by Ruehl et al. [23]. This approach can systematically show variability points and their relationships. This work focuses on the creation of descriptions of variability but not so much on the execution. Weissbach and Zimmermann [24] tackle the problem of avoiding storing or processing data at undesired location by data-flow analysis. In contrast to our work, this approach is not context-aware. There are also numerous works on context-aware service oriented systems. Du et al. [25] controls data-flow between services to detect malicious services. Context awareness with respect to the client is not assumed. Azeez et al. [26] propose a multi-tenant service-oriented architecture middleware for cloud computing. They;concentrate on multiple users sharing an instance and native multi-tenancy. Contrary to our work, using certain services in context of the location is not considered. Bastida et al. [27] discuss the steps that the service integrators should follow to create context-aware service compositions and also introduce a composition platform that supports the lifecycle of dynamic compositions both at design-time and at runtime. The context part is not explicitly defined in the complete approach.

Table I shows a comparison among several research works in the area of management and configuration of cloud environments. In the state of the art, some work has been performed to combine the benefits SPLE with those of multi-tenancy to facilitate the customization of SaaS applications tailored to the tenant-specific needs. However, none of the current approaches defines explicitly the context-aware of the tenants and users in the complete approach in both design time and run time phase (see Table I). Moreover, it provides no support for context awareness which is one of the keystones for the cloud computing in general and SaaS in particular.

TABLE I. A Comparison among research works on Cloud Environment

Research work	Adaptation Type	Phase of system variability	Adaptation Space	Adaptation Mechanisms	Environment
[20]	Dynamic	Design time	Functional	Variability	SaaS
[21]	On-demand	Design time	Functional and non-Functional	Variability	SaaS
[22]	Dynamic	Design time	Functional and non-functional	Variability	SaaS
[23]	Dynamic	Run time	Non-Functional	Variability	Data security in the cloud
[24]	Dynamic	Run time	Non-functional	Variability	IaaS
[25]	Static	Design time and Run time	Functional	variability	Middleware
[26]	Dynamic	Design time and Run time	Functional and context-aware	variability	Composants

VII. CONCLUSION AND FUTURE WORK

This paper presented our first steps towards autonomic and dynamic context-aware configuration variability on the SaaS applications. We identified requirements for a multi-tenant aware SaaS reference architecture at design time as well as at runtime. In addition, we have shown an overview of our process which our framework will be based. We rely on autonomic system concept in order to allow a dynamic and automatic management of variability for these applications. Furthermore, our dynamic configuration process allows deriving multiple variant configurations that are independent from each other.

Because SPL engineering is a well researched field, we may benefit from developed tools that help to derive valid tenant configurations and we propose to use NSGA-II (Non-Dominated Sorting Genetic Algorithm) algorithms [28] to optimize and select services. Additionally, we plan to take into account context user's evolution. As the cloud market evolves constantly, changes in context can occur that require the application environments to be reconfigured. e.g., a new service is available. To deal with such changes, we propose to adapt evolutionary tree and evolutionary algorithm.

REFERENCES

[1] P. Mell and T. Grance, "The nist definition of cloud computing," NIST Special Publication 800-145, National Institute of Standards and Technology, Information Technology Laboratory, Sept, 2011, pp. 3-8.

[2] G. F. Chong and G. Carraro, "Architecture strategies for catching the long tail," Website, April, 2006, pp. 10-26. [retrieved: 08, 2014]. Available: http://msdn.microsoft.com/en-us/library/aa479069.aspx

[3] M. L. Griss, "Implementing product-line features with component reuse," in Proceedings of the 6th International Conerence on Software Reuse: Advances in Software Reusability, London, UK, June, 2000, pp. 137–152.

[4] D. M. Weiss and C. T. R. Lai, "Software product-line engineering: a family-based software development process," Addison-Wesley Professional, Aug, 1999, 448 pages.

[5] A. Benali and B. El Asri, "Towards dynamic management of variability and configuration of cloud Environments," in press

[6] S. Barney, A. Aurum, and C. Wohlin, "A product management challenge: creating software product value through requirements selection," Journal of Systems Architecture, vol. 54, no 6, June, 2008, pp. 576-593.

[7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Technical report, CMU/SEI TR-21, USA, Nov, 1990, 148 pages.

[8] L. Cardelli, "Program fragments, linking, and modularization," In Proc. Symp. Principles of Programming Languages (POPL), ACM Press, Mar, 1997, pp. 266–277.

[9] I. Trummer, "Cost-optimal provisioning of cloud applications," Diploma thesis, University of Stuttgart, Faculty of computer science, electrical engineering and information technology, Germany, Feb, 2010, pp. 135 – 142.

[10] B. Schilit and M. Theimer, "Disseminating active map information to mobile hosts," IEEE Network, 8(5), Sept, 1994, pp. 22-32.

[11] A. Dey and G. Abowd, "Towards a better understanding of context and context-awareness," in CHI 2000 Workshop on The What, Who, Where,When, and How of Context-Awareness, nov, 2001, pp. 304-307.

[12] R. Want, A. Hopper, V. Falcao, and J. Gibbons, "The active badge location system," ACM TIS, Jan, 1992, pp. 91-102.

[13] R. Hull, P. Neaves, and J. Bedford-Roberts, "Towards situated computing," International Symposium on Wearable Computers, Oct, 1997, pp. 146-153.

[14] K. Dey and D. Abowd, "Towards a better understanding of context and context-awareness," Georgia Institute of Technology, Atlanta, GA, USA 30332-0280, Sept, 1999, pp. 271-350.

[15] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker, "Generative programming for embedded software: an industrial experience report," In: D.Batory, C.Consel, W.Taha, GPCE 2002. LNCS, vol. 2487, Oct, 2002, Springer, Heidelberg (2002), pp. 156–172.

[16] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration through specialization and multi-level configuration of feature models," Improvement and Practice Journal, April, 2005, pp. 143-169.

[17] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," Computer, IEEE, Jan, 2003, pp. 41-50.

[18] D. Martin, M. Burstein, J. Hobbs, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic markup for web services," Website, Nov, 2004. [retrieved: 08, 2014].Available: http://www.w3.org/Submission/OWL-S/

[19] J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau, "Dynamic configuration management of cloud-based applications," In: SPLC '12:16th International Software Product Line Conference – Vol. 2, ACM, pp. 171–178.

[20] C. Cetina, P. Giner, J. Fons, and V. Pelechano, "Autonomic computing through reuse of variability models at run-time: the case of smart homes," IEEE Computer Society Press, Los Alamitos, CA (2009), Oct, 2009, pp. 37-43.

[21] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl, "Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications," In (PESOS '09) Proceedings, USA, May, 2009, pp. 18-25.

[22] X. Cheng, Y. Shi, and Q. Li, " A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on SLA," In Proceedings of the Joint Conferences on Pervasive Computing, JCPC '09, Dec, 2009, pp. 599-604.

[23] S. T. Ruehl and U. Andelfinger, "Applying software product lines to create customizable software-as-a-service applications," In Proceedings of the 15th International SPL Conference, Volume 2, ACM, Aug, 2011, pp. 16:1-16:4.

[24] M. Weissbach and W. Zimmermann, "Controlling data-flow in the cloud," in The Third International Conference on Cloud Computing, GRIDs, and Virtualization, W. Zimmermann, Y. W. Lee, and Y. Demchenko, Eds. ThinkMind, July, 2012, pp. 24–29.

[25] J. Du, W. Wei, X. Gu, and T. Yu, "Runtest: assuring integrity of data flow processing in cloud computing infrastructures," in Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ser. ASIACCS '10. New York, NY, USA: ACM, Jan, 2010, pp. 293–304. [retrieved: 08, 2014]. Available: http://doi.acm.org/10.1145/1755688.1755724.

[26] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle, "Multi-tenant soa middleware for cloud computing," in IEEE CLOUD, July, 2010, pp. 458–465.

[27] L. Bastida, F. J. Nieto, and R. Tola, "Context-aware service composition: a methodology and a case study," In SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments, New York, NY, USA, ACM, May, 2008, pp. 19–24.

[28] A. Pratap, T. M. K. Deb, and S. Agrawal, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization : NSGA-II," Technical report, Indian Institute of Technology Kanpur, Sep, 2000, pp. 849-858.

.