Journal of Software Engineering and Applications, 2012, 5, 536-544
http://dx.doi.org/10.4236/jsea.2012.58062 Published Online August 2012 (http://www.SciRP.org/journal/jsea)

Understanding Requirement Engineering for Context-Aware Service-Based Applications

Seyed Hossein Siadat, Minseok Song

School of Technology Management, Ulsan National Institute of Science and Technology, Ulsan, South Korea.
Email: siadat@unist.ac.kr, msong@unist.ac.kr

Received May 3rd, 2012; revised June 1st, 2012; accepted June 12th, 2012

ABSTRACT

Requirements of software systems tend to change over time. The speed of this tendency depends on the application domain the software system under consideration belongs to. If we consider novel contexts such as pervasive systems and systems supporting dynamic B2B interaction, requirements change so fast that the research community is studying how to build systems that are able to self-adapt on the fly to some of these changes. When this happens, the system does not need to undergo through a new development cycle thus increasing its availability and, to a certain extent, its robustness. So far, the research in the area of self-adaptive systems has been focusing on the definition of the mechanisms for supporting self-adaptation. We argue that what is missing now is a structured and robust design process associated to these mechanisms. This design process should include a Requirement Engineering (RE) phase that somewhat differs from the traditional one. However, the identification of requirements for adaptation requires a good knowledge of the context in which the system will be executed. In this work, we consider the modeling of such context as part of the RE phase and we particularly focus on Service-Based Applications (SBAs). We argue that RE activities should be supported at runtime to handle context changes and to support adaptation for SBAs. We survey the state of the art for what concerns the elicitation, modeling, and analysis of requirements and will highlight some issues and challenges in order to support adaptation for SBAs.

Keywords: Requirement Engineering; Service-Based Applications; Dynamic Adaptive Systems; Adaptation; Context

1. Introduction

Classical requirement engineering is based on the assumption that the environmental context is static and can be understood sufficiently. In the presence of these two factors of environment namely being static and well-understood, traditional requirement engineering could be performed well. This might be still possible due to the speed of change tendency. This means, where such context changes slowly enough, existing techniques are capable of capturing, managing and adapting these contextual changes. However, if we consider novel contexts such as pervasive systems and systems supporting dynamic B2B interaction, requirements change so fast that the research community is studying how to build systems that are able to self-adapt on the fly to some of these changes. The dynamicity and uncertainty of the environmental context are the main two obstacles in understanding requirement for adaptive systems. These make it difficult to understand, discover, formulate, validate, reason and manage the requirement both at design and specially at runtime.

This design process should include a requirement engineering phase that somewhat differs from the traditional one. Also, the identification of these requirements for adaptation requires a good knowledge of the context in which the system will be executed. At one side, we argue that what is missing now is a structured and robust design process for RE. However, the design time decisions need to be done in the situation of incomplete and uncertain knowledge about environmental context. This way, we need to understand to what extent the requirements are being satisfied and this can support adaptation strategies at run-time. Therefore, at the other side, we argue that requirements for adaptive systems should be supported not only at design-time but also at run-time. The ability of an adaptive system to be a "requirement-aware" system inspired by [1], in which the authors argue that requirements for self-adaptive systems should be run-time entities that could be reasoned over at run-time.

Therefore, the need for some methods and techniques to develop some kinds of systems capable of automatically reconfiguring and dealing with any faults and attacks is vital. Research on self-adaptive system is a response to this need, and results in systems able to detect any internal faults and environmental changes and accordingly adapt its structure and behavior [2].

This work focuses on requirements engineering, as a basic discipline in developing each system, and aims at presenting the state of the art of this fundamental discipline for self-adaptive systems and in particular SBAs.

In realizing the adaptive behavior of SBAs, the role of the context is very important. Requirements modeled at design time can be vary over the time, therefore, they may not be satisfiable when the context changes. The term context may vary from different perspective as different literatures define various definitions and elements for context. We argue about our definition of context and classify different elements of it. It is necessary to define a context model as part of the requirement engineering for SBAs. Such context model provides information for triggering situations for the adaptation of SBAs.

We explain some key challenges in supporting requirement engineering techniques for dynamic systems in presence of volatile and uncertain environmental context. We argue that requirement engineering activities should be supported at run-time to handle requirements for dynamic systems.

The remainder of the paper is organized as follows. We start by an introduction to self-adaptive systems and their major features in Section 2. In Section 3, we explain an extended life-cycle for adaptive SBAs in general and the phases and corresponding activities that need to be supported for each phase. Section 4 discusses about SBAs as a dynamic system and the fact that adaptation need to be supported in such systems. A context information model to support adaptation is described as well. An overview on the main challenges in RE for adaptive service-base systems is reported in Section 5. We conclude the paper in Section 6.

2. Dynamic Adaptive Systems

The propagation of dynamic adaptive systems in various fields has provided opportunity in conducting research in different development phases from preliminary analysis to implementation. Such systems have been applied in autonomic computing, pervasive systems, ubiquitous computing and service-oriented computing. Although there are some definitions for self-adaptive systems, the existing concepts in various domains such as pervasive systems and autonomic computing need to be clarified. In order to explain the boundary of self-adaptive systems, we present some well-known definitions and characteristics self-adaptive systems should support.

DARPA Broad Agency Announcement (BAA) [3] presents the following definition for self-adaptive systems: "Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality and performance is possible". Firstly, the definition expresses that software systems have several ways of realizing their functionalities. Second, software systems should have adequate knowledge of their structure in order to make suitable changes at runtime. Therefore, the software should be able to evaluate its behavior, and re-plan its operations. Indeed, a typical self-adaptive system uses a closed loop like MAPE (Monitoring, Analysis, Planning, Execution) similar to the one used in autonomic computing [4]. Dobson *et al.* [5] represented these four activities in a similar approach with a closed control loop: collect, analyze, decide and act.

Oreizy *et al.* [6] present another definition: "Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation". The term *operating environment* is stressed in this definition. Furthermore, the fact that system needs to be aware of its environmental changes is underlined. Therefore, one main characteristic of self-adaptive systems is checking continuously for possible changes in their operating environment as well as internal elements. The system is required to respond to such changes to satisfy its main goals.

The above properties are named "context-awareness" and "self-awareness" by Salehi *et al.* [2], as primitive characteristics of self-adaptive systems that need to be supported to acquire a minimum degree of self-adaptiveness. In addition, a multi-level pyramid including three levels of self-* properties is addressed in [2]. The proposed pyramid introduces self-adaptiveness as a general property that can be separated into major and primitive properties. The primitive properties are required to be supported by the system such that no adaptation is achieved without realizing a minimum level of primitive properties. The major properties are those properties addressed by autonomic computing [7] *i.e.* self-configuring, self-healing, self-optimizing, and self-protecting. In fact, these properties represent various adaptations each system need to aware of.

3. An Extended Framework for Adaptation Life Cycle in SBAs

In this section we explain the S-cube life-cycle for adaptive SBAs [8]. The life-cycle is represented in **Figure 1**. It covers both cycles of design-time and run-time such that they coexist and support each other. The design cycle is specified for permanent and important change while the run-time cycle is for temporarily adaptation of

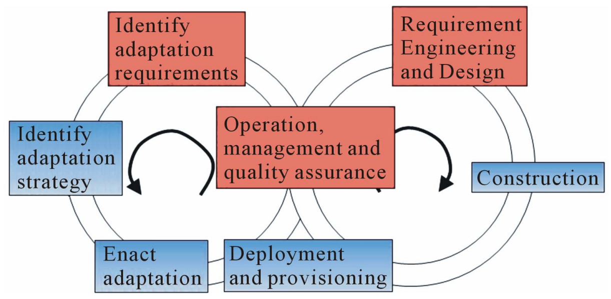
Identify adaptation requirements
Requirement Engineering and Design
Identify adaptation strategy
Operation, management and quality assurance
Construction
Enact adaptation
Deployment and provisioning

Figure 1. Adaptation life-cycle.

the SBAs. Different phases are involved: requirement engineering and design, construction, deployment and provisioning, operation and management, identify adaptation needs, identify adaptation strategy and enact adaptation. For the rest of the work we only focus on the phases that are shown in the **Figure 1** by Red color.

At the requirement engineering and design phase the adaptation and monitoring requirement are used to perform the design for adaptation and monitoring. The run-time monitoring is executed at the operation and management phase. Context changes of SBAs are detected in this phase. Context information captured from the monitoring provides the adaptation triggers. Such triggers identify adaptation need at the next phase. Each adaptation need can be satisfy be a set of adaptation strategies. In the following we discuss about issues need to be incorporated into the framework in order to support adaptation in SBAs.

Context aware systems are capable to detect changes and are able to change their behavior to adapt to the changing context. In such systems, changes are not only performed by users by also other sources are involved. Therefore, good understanding of the context is necessary. Furthermore, there is a need for user and context modeling in the RE design phase. This should be done through precise context engineering, identifying different context elements and their dependencies. Distinguishing between stable and non-stable context is important and useful for the decision phase. With respect to adaptation, the needs and strategies for adaptation should be identified. Model-based RE such as scenario-based approaches could be applied in order to link context information to adaptation strategies. Scenario-based approaches are useful for the development of systems when the context changes are predictable or at least have a low degree of uncertainty. Additionally, user modeling techniques are useful to present the participant aspect of the usage context. The so far activities should be done at the requirement and engineering design phase through a requirement elicitation and modeling.

After this, requirement engineering is to support which adaptation need to be done given a situation. At the operation and management phase, the context changes should be monitored and detected. Understanding the degree and scope of change and uncertainty level are important and help to come up with the right decision for adaptation. This information provides triggers for the next phase to define adaptation requirements. It should be identified whether the adaptation is going to be automatic or semi-automatic. As in the case of semi-automatic, the user may involve in the process of adaptation decision. This can be done by providing appropriate feedback to users.

The characteristic of context aware systems bring the need to elicit, model and monitor requirement for such systems. Thus we discuss requirement engineering activities and corresponding techniques to support aforementioned issues. Besides, we provide a context information model to support the adaptation of SBAs.

4. Context-Aware Systems and SBAs

Context-aware SBAs are required to be aware of the context, sense the environment, detect context changes and act accordingly. The main issue here is the relationship and linking between the environmental behavior (context) and the system behavior (requirement). The state of the environment and consequently its context will have a major effect on such relation. If the context is well-understood and stable then the appropriate adaptation actions could be define perfectly and clearly at the design time. However, where the context is not well-understood and not-stable then such relation is not clear and it is hard to make decision. This is mainly due to the uncertainty aspect of the context. Therefore it is necessary to monitor, detect and analyze the context at run-time when the system is deviating from the early requirements. In this situation, adaptation decisions made at design time are not adequate and therefore new decisions need to be made at run-time according to the information of context changes. This is even more difficult when considering non-functional requirements *i.e.* QoS issues in Web Services.

Requirement engineering for adaptive systems and in particular for service-based applications can be categorized into three parts: requirements elicitation, requirements modeling and specification, and finally requirements monitoring. In the following we present an overview on related work discussing main contributions in each part.

Requirements Elicitation: includes activities to identify stakeholders, goals, and requirements in general [9]. Regarding self-adaptive systems, requirements are dependent on the contexts the software system under consideration belongs to. Therefore, self-adaptive systems have to adapt their behavior according to context changes. For such purpose, applying context engineering during requirement elicitation can be beneficial.

A contextual requirements discovery for ubiquitous systems is proposed in [10]. The linking between requirements scenarios and context information is discussed in the work and requirements analysts are supported by contextual tools [10,11]. This way, analysts are able to observe run-time environmental changes and events in order to discover new requirements for evolving system. For such purpose, data mining techniques have to be taken into account [12]. Other approaches [5, 13] for contextual and personal requirement engineering have been proposed in the literature to extract stakeholders' requirement.

Requirement Modeling: Goal-oriented requirement engineering approaches have been mainly considered as a key solution for requirements modeling and specification in adaptive systems [14,15]. Stakeholders' goals and system objectives are relatively stable [16] whereas requirements define one of the possible ways that a goal can be realized which means goals are operationalized through requirements [17].

Goal-oriented approaches (*i.e*. KAOS [17], i* [18] and Tropos [19]) allow analysts to obtain and define goals as well as requirements. They are also able to identify constraints that environment enforces to requirements. Hierarchical goal models and refinement approaches can be used for adaptation techniques and developing of adaptive systems in which they allow analysts to describe various contextual requirements in order to achieve a goal.

Since the matter of time is a key factor in adaptive systems, therefore Linear Temporal Logic (LTL) has been applied in goal-oriented approaches [14,17,20]. In addition, event modeling has a key part in adaptation modeling. For example, [21] addresses an event modeling approach for adaptation modeling by proposing an XML-base rule modeling for providing more flexibility to system design in dealing with possible changes. However, current goal-oriented approaches cannot completely cope with the challenges of RE for self-adaptive systems [14, 22].

Requirement Monitoring: In order to ensure that the requirements are properly fulfilled, self-adaptive systems need to be able to monitor the environment. [23] argues that requirements as well as designs issues are typically formulated in a set of assumptions about the context. Therefore, requirements specification and system design are based on a set of assumptions which their stability cannot be guaranteed.

For that purpose, an approach using event-based formal language (called FLEA) is developed to give users the ability to monitor functional requirements and assumptions on-the-fly [24]. The approach is able to describe the conditions required for executing an adaptation. Moreover, [25] integrates FLEA approach with a goal-oriented approach to identify requirements deviation at run-time.

Nevertheless, classical requirement engineering techniques [9,10,14,15,17,20,21,23-27] are not adequate to support the adaptive behavior in context-aware SBAs. Most of the current approaches do not consider adaptive behavior of the system. Although the approaches are automatic but they do not support run-time adaptation and in most cases after detecting any violation, the requirement engineering starts from design time. An approach for run-time monitoring and adaptation of web services works based on feedback control loop is proposed in [28]. Using the feedback the current requirement model will be updated and analyzed to detect violation. Consequently automatic reconfiguration is done to perform recovery actions.

Regarding requirements monitoring at run-time, [29] proposes an approach (called ReqMon) to monitor requirements satisfaction in information systems. The approach uses an event-based framework that accepts goal-based requirements formalized by LTL, and afterwards generates a monitor code, which eventually makes a relation between high-level goals and low-level run-time events. Monitoring of functional and non-functional requirements in the context of service-oriented architecture is addressed in [30] so that binding between service consumer and provider can be changed dynamically over time.

As we presented above, some significant approaches have been addressed to support RE activities for adaptive systems. However, the research in this area has still much to do and is in its beginning steps. Besides, these approaches are dealing with a specific activity at one time and therefore they are isolated approaches and there is a need to provide a comprehensive framework that incorporates complete RE activities in one approach. In the following we argue about some initial work trying to provide a comprehensive approach including RE activities and adaptation decisions with respect to the context changing.

A framework of RE in context-aware services is proposed in [16]. A reflection-based framework is presented for such purpose. The framework address issues such as changing context and changing requirement. In their approach the context changes include: *changing location*, *changing bandwidth*, *changing display characteristics* such as graphic PDAs or text-only mobile phones, *changing usage paradigm* and the last one that the *target platform is unknown in advance*.

RE techniques for context aware systems are proposed in [2]. The authors presented a model-based requirement engineering approach named RE-CAWAR to analyze the basic system and adaptation behavior of the context aware systems. In their approach the context changes include: *changing participants* such as changing location and personal properties (e.g. age and education), *changing activities* that indicate tasks and goals of participants influenced by environmental events, and finally *changing operational environment* such as network conditions and physical factors (e.g. temperature, light, humidity). [31] proposes PC-RE (Personal and Contextual Requirements Engineering) method that allows requirements to change over the time in presence of contextual uncertainty. A scenario-based approach is described to specify the requirements and their changes.

A common limitation of above approaches is a lack of proper context model that provides information for adaptation decisions. The requirement that identified at design-time, may not be satisfiable when the context changes. This can affect the performance of SBAs. However, context is a very broad term and understanding it requires a special care. Different elements of the context need to be accurately classified. Moreover, dependencies between context elements need to be identified in order to prevent propagation of changes from one context element to the other one. In the following we first of all present our definition of the term context and then classify context elements into different categories.

Context Classification

Here we present our definition of context as such: "Context" is any information that influences the interaction between users and a service-based application.

We classify context elements into six distinct categories: resource, user, provider, environment, web service quality and web service functionality. This provides us a comprehensive view of information that influences Service-base applications. The context information model drives situation that triggers adaptation. Context classification is illustrated in **Figure 2**. These elements are subject to change during the life-cycle of SBAs. We explain each of them in the following briefly.

Resource Context: It includes hardware and software properties that influence SBAs. Availability of the resources has an impact on satisfying the requirement. The information of resources and their availability could be updated during changes. The resource context also contains characteristics of network and operating systems for accessing the SBA.

User Context: The user context includes the user's requirement and preferences. Requirement priorities from user perspective are expressed in this category. For example, regarding QoS requirement it shows which properties will be maximized among others. It also contains the information about the role of the user in the application e.g. guest or administrator.

Provider Context: It covers information from the provider side on the usage of the SBA. Provider may change the offered requirement during the execution. For example the provider may increase or decrease the computational charge and this will have a direct impact on the perceived requirement of SBA from the used side.

Environment Context: It has information about the time in which users access the SBA or the information about where the user is located. It also covers the surrounding environment such as the current date, temperature and weather. The modification of this context is performed by either users or external events.

Web Service Quality Context: It covers information about non-functional properties of web services in SBAs. Typical non-functional properties include availability, throughput, response time, level of security and they are often collectively referred to as quality dimensions. Changes of other contexts have mostly direct impact on the quality context. However, any changes in the quality context will trigger adaptation.

Web Service Functional Context: It contains information about functional properties of web services in SBAs regarding interface, structure of messages and different protocols. The cause of a functional change could be performed by changing requirement from user, provider and even environment contexts. Consequently there is a need to add, remove or update functionality. For instance, changes in the physical environment (e.g., temperature) can influence the network characteristic and the quality context (e.g. response time) also change. This will triggers an adaptation situation and the functional context is required to be updated.

5. Challenges and Discussions

In this section, we discuss and highlight some issues and challenges in order to support adaptation for SBAs.

Requirement Reflection: Requirements of adaptive systems need to be represented at run-time (run-time entities) to support adaptation. This involves modeling requirements at design time and reasoning them at run-time according to changing context to support adaptation. Therefore, the selection of the best adaptation strategy will be postpone at run-time by reasoning the existing requirements model and run-time data acquired from the context changes. This way, it is possible to revise and re-evaluate design-time decisions at run-time.

Goal-oriented approaches [17-19] seem to be a promising method for supporting requirement reflection. For example KAOS [17] provides a modeling language with formal semantics that allows automated reasoning over requirements and goals.

Dynamicity of Requirements: Due to the dynamicity of the requirements, it is necessary to specify the evolution of the requirement model [1]. There are some challenges and research issues that need to be taken into account when the requirement model changes at run-time.

At one hand, the requirement model itself should be supported with model transformation patterns. With this regards, a library of requirements model transformation operators is proposed by [32]. Examples of such operators are: add-requirement, delete-requirement, replace-requirement, add-goal, delete-goal, replace-goal and so on. On the other hand, these transformation models and requirement evolution models need to be synchronized with other software components, particularly software architecture. Coulson *et al.* [33] proposed an approach that supports such synchronization by introducing reflective architectures. The approach is composed of two layers, namely a base layer which include the actual running architecture and a meta layer that is responsible for dynamically managing the running architecture. A similar approach is proposed in [1] which the authors discuss a semantic integration between the requirements and architecture models.

A process called LoREM is proposed in [34] for handling requirement engineering activities, where each level describes requirement engineering activities of different developers in a dynamic adaptive system. These levels are motivated from four levels of requirement engineering proposed in [26]. Requirements, goals and system functionality are represented in level 1, adaptation scenarios are identified in level 2 and adaptation infrastructure is presented and configured in level 3 in order to provide adaptation scenarios. The relationship between levels is supported through a model-driven development.

Uncertainty of Requirements: Uncertainty is a fundamental issue and a major challenge in almost all intelligent systems. Theories of uncertainty have been identified in management and economics. Such theories could be application for self-adaptive software systems. So far there is a lack of such theories in dealing with uncertainty in requirement engineering models. In order to deal with uncertainty, we need to be able to represent/ model it and reason about it. Various Techniques and frameworks have been introduced for reasoning uncertainly [35]. Apart from this, understanding the degree of uncertainty of the context is necessary. Classification of uncertainty degree is reported in the literatures [36].

For example, consider a situation that there are different possible future scenarios but it is possible to list them all. Considering adaptation scenarios, this requires all possible scenarios to be taken into account at the design time. This could be done by considering all alternative contextual conditions and design all adaptation scenarios based on them. Therefore, run-time decision will be mainly based on requirement engineering at design time. In general, as long as the changes in the context are known, run-time decisions could be handle using existing requirement engineering techniques such as defining adaptation trigger conditions. Now, consider a situation that it is feasible to construct future scenarios but these are mere possibilities and are unlikely to be exclusive. This is a situation that context changes cannot be anticipated. RE at design time would not support run-time decisions. Defining adaptation trigger conditions at design-time is not adequate any longer as the new triggering conditions cannot be predicted.

In order to overcome such challenges we argue that two consideration need to be taken into account. First is a move from binary satisfaction of the requirement. Degree of the requirement satisfaction need to be evaluated (e.g. using a fuzzy approach) and corresponding adaptation actions should be selected accordingly. Second is defining critical and non-critical requirement. Therefore we can distinguish between vital and trivial requirement. For example, it is possible to temporarily ignore some requirement with non-criticality in favor of other critical requirement (requirement trade-off approach).

Dealing with uncertainty has been recently treated as a hot issue in the literatures [37]. Initial solutions for overcoming uncertainty limitations are reported in [38] which later resulted in development of a new language named RELAX [39]. It provides the system with the flexibility to trade-off the requirements at run-time and allows some certain requirement to be temporarily RELAXed.

Adaptation Strategies and Decision Making: A range of available adaptations (strategies) could be identified at the design time. Requirements obtained from various context information models (see **Figure 2**) can identify triggering conditions for adaptation. Then the changes of the context can be linked to the adaptation strategies by identifying rules. Therefore, finding most suitable adaptation strategies (between the alternatives) will be done at run-time. However, the two aforementioned challenges namely dynamicity and uncertainty of the requirements make the adaptation decisions to be unpredictable.

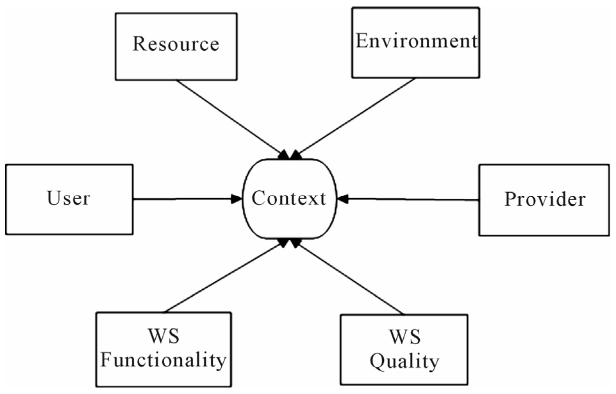

Figure 2. Categorization of context information.

In this situation, decisions have to be evaluated. Besides, each adaptation strategy has a different trade-off and consequences that has to be analyzed with the information at run-time. Multi-objective decision making may be applicable when the uncertainty exists. It usually defines a utility function that calculates the weighted sum of different objectives. However, regarding adaptation strategies this can be a difficult task identifying the weight of each strategy.

There are issues that need to be taken into account as following. First of all, the monitoring data should be used to evaluate the context properties identified in the context model. Therefore the context changes need to be detected and the degree of changes need to be evaluated. Understanding uncertainty level as we explained earlier is also necessary. The aggregation of this consideration will result to identify the adaptation triggers. The triggers are the base to define adaptation needs. The existing rules and links between the context and adaptation strategies need to be updated according to the information obtained at monitoring. Apart from these, the user preferences could mainly affect choosing the right adaptation. Furthermore, adaptation purpose need to be identified in the early stage whether is it for optimization, recovery or prevention as each may have different requirements.

Feedback Loop: From Control Theory to Software Engineering: The notion of feedback loop has been widely used in the field of control engineering. Actually the control loop is recognized as the central element of control theory. [5] presents a generic model of a control loop that includes four key activities namely: collect, analyze, decide and act. Cheng *et al.* [40] upgraded the generic model by identifying properties of control for each activity which were ignored in the generic model. For example in the analyzing part, we need to know how much past state may be needed in the future. In the decision part, we need to know how the future state of the system is inferred. Or what are the priorities for adaptation across multiple control loops. And finally in the last part, action part, we need to know when the adaptation should be performed.

Applicability of using control theories for self-adaptive systems is still under investigation. However, in order to address changes in the context, borrowing theories from control engineering and apply them from self-adaptive systems could be beneficial. [41] identifies engineering principles for self-adaptive systems through feedback loop. Discussions about the control loops should be an explicit activity is reported in [42]. There is a similar discussion in [43] about the need of the control loop to be explicit. Additionally the authors argue that one explicit loop in not enough and in order to support various changes the system is required to have different nested loop.

There are some work particularly uses feedback loop for adaptation in SBAs. [28] uses a feedback loop at runtime to handle both functional and non-functional properties of services. The feedback loop is used to update the requirement model and analyze it at runtime which make it possible to detect the violations and perform the automatically recovery action in order to guarantee the system goals. An explicit feedback loop technique for the adaptation of complex service oriented systems is used in [43].

6. Conclusion

We discussed the state of the art of requirements engineering for adaptive systems. We started by briefly describing significant characteristics of self-adaptive systems and continued by explaining the adaptation lifecycle in SBAs. With this regard, we discussed about corresponding activities and issues need to be incorporated into the framework in each phase. We focused on requirements engineering activities namely requirements elicitation, requirements modeling and specification, and requirements monitoring, as a basic discipline in developing adaptive systems and in particular for SBAs. We argued about our definition of context and classified different elements of it. Moreover, we pointed out the importance of defining a context model as part of the requirement engineering for SBAs. Such context model provides information for triggering situations for the adaptation of SBAs.

7. Acknowledgements

"This work was partially supported by the Industrial Strategic technology development program (10040142, Development of a qualitative customer feedback analysis and evaluation method for effective performance management in B2C Industry) funded by the Ministry of Knowledge Economy (MKE, Korea)". In addition, the initial part of the work was done at Politecnico di Milano and the first author would like to thank Dr. Elisabetta di Nitto for her supervision.

REFERENCES

[1] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein and E. Letier, "Requirements Reflection: Requirements as Runtime Entities," *International Conference on Software Engineering*, Capetown, 2-8 May 2010, pp. 199-202.

[2] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 4, No. 2, 2009, pp. 1-42. doi:10.1145/1516533.1516538

[3] R. Laddaga, "Self Adaptive Software Problems and Projects," *The 2nd International IEEE Workshop on Software Evolvability*, IEEE Computer Society, Washington,

2006, pp. 3-10.

[4] IBM, "An Architectural Blueprint for Autonomic Computing," *IBM White Paper*, 2005.

[5] S. Dobson, S. Denazis, A. Fernandez, D. Gaıti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt and F. Zambonelli, "A Survey of Autonomic Communications," *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 1, No. 2, 2006, pp. 223-259. doi:10.1145/1186778.1186782

[6] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum and A. L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software," *IEEE Intelligent Systems*, Vol. 14, No. 3, 1999, pp. 54-62. doi:10.1109/5254.769885

[7] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer*, Vol. 36, No. 1, 2003, pp. 41-50. doi:10.1109/MC.2003.1160055

[8] A. Bucchiarone, R. Kazhamiakin, C. Cappiello, E. DiNitto and V. Mazza, "A Context-Driven Adaptation Process for Service-Based Applications," *The* 2*nd International Workshop on Principles of Engineering Service-Oriented Systems*, ACM, New York, 2010, pp. 50-56.

[9] B. H. C. Cheng and J. M. Atlee, "Research Directions in Requirements Engineering," *Proceedings of Future of Software Engineering*, IEEE Computer Society, Washington, 2007, pp. 285-303.

[10] N. Seyff, F. Graf, P. Grunbacher and N. Maiden, "Mobile Discovery of Requirements for Context-Aware Systems," *The* 14*th International Conference on Requirements Engineering: Foundation for Software Quality*, Springer-Verlag, Berlin, 2008, pp. 183-197.

[11] N. Maiden, N. Seyff and P. Grunbacher, "The Mobile Scenario Presenter: Integrating Contextual Inquiry and Structured Walkthroughs," *The* 13*th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE Computer Society, Washington, 2004, pp. 115-120. doi:10.1109/ENABL.2004.65

[12] J. Cleland-Huang and B. Mobasher, "Using Data Mining and Recommender Systems to Scale up the Requirements Process," *The* 2*nd International Workshop on Ultra-Large-Scale Software-Intensive Systems*, ACM, New York, 2008, pp. 3-6.

[13] T. Cohene and S. Easterbrook, "Contextual Risk Analysis for Interview Design," *The* 13*th IEEE International Conference on Requirements Engineering*, IEEE Computer Society, Washington, 2005, pp. 95-104. doi:10.1109/RE.2005.20

[14] G. Brown, B. H. C. Cheng, H. Goldsby and J. Zhang, "Goal-Oriented Specification of Adaptation Requirements Engineering in Adaptive Systems," *Proceedings of the* 2006 *International Workshop on Self-Adaptation and Self-Managing Systems*, Shanghai, 20-28 May 2006, pp. 23-29. doi:10.1145/1137677.1137682

[15] H. Nakagawa, A. Ohsuga and S. Honiden, "Constructing Self-Adaptive Systems Using a Kaos Model," *Proceedings of the* 2008 2*nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, IEEE Computer Society, Washington, 2008, pp. 132-137.

[16] A. Finkelstein and A. Savigni, "A Framework for Requirements Engineering for Context-Aware Services," *Proceedings of* 1*st International Workshop from Software Requirements to Architectures*, Toronto, 14 May 2001.

[17] A. Dardenne, A. Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition," *Science of Computer Programming*, Vol. 20, No. 1-2, 1993, pp. 3-50. doi:10.1016/0167-6423(93)90021-G

[18] E. Yu. "Modelling Strategic Relationships for Process Reengineering," Ph.D. Thesis, University of Toronto, Toronto, 1995.

[19] P. Giorgini, M. Kolp, J. Mylopoulos and M. Pistore, "The Tropos Methodology: An Overview," *Methodologies and Software Engineering for Agent Systems*, Kluwer Academic Press, New York, 2003, pp. 505-525.

[20] J. Zhang and B. H. C. Cheng, "Using Temporal Logic to Specify Adaptive Program Semantics," *Journal of Systems and Software*, Vol. 79, No. 10, 2006, pp. 1361-1369. doi:10.1016/j.jss.2006.02.062

[21] Q. Wang, "Towards a Rule Model for Self-Adaptive Software," *SIGSOFT Software Engineering Notes*, Vol. 30, No. 1, 2005, pp. 8-12. doi:10.1145/1039174.1039198

[22] W. Sitou and B. Spanfelner, "Towards Requirements Engineering for Context Adaptive Systems," *The* 31*st Annual International Computer Software and Applications Conference*, IEEE Computer Society, Washington, 2007, pp. 593-600.

[23] S. Fickas and M. S. Feather, "Requirements Monitoring in Dynamic Environments," *The* 2*nd IEEE International Symposium on Requirements Engineering*, IEEE Computer Society, Washington, 1995, pp. 140-147.

[24] D. Cohen, M. S. Feather, K. Narayanaswamy and S. S. Fickas, "Automatic Monitoring of Software Requirements," *The* 19*th International Conference on Software engineering*, ACM, New York, 1997, pp. 602-603.

[25] M. S. Feather, S. Fickas, A. Van Lamsweerde and C. Ponsard, "Reconciling System Requirements and Runtime Behavior," *The* 9*th International Workshop on Software Specification and Design*, IEEE Computer Society, Washington, 1998, pp. 50-59.

[26] D. M. Berry, B. H. C. Cheng and J. Zhang, "The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems," *The* 11*th International Workshop on Requirements Engineering Foundation for Software Quality*, Porto, 13-14 June 2005, pp. 95-100.

[27] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 4, No. 2, 2009, pp. 1-42. doi:10.1145/1516533.1516538

[28] I. Epifani, C. Ghezzi, R. Mirandola and G. Tamburrelli, "Model Evolution by Run-Time Parameter Adaptation," *The* 31*st International Conference on Software Engineering*, IEEE Computer Society, Washington, 2009, pp. 111-121.

[29] N. Robinson, "A Requirements Monitoring Framework for Enterprise Systems," *Requirements Engineering*, Vol. 11, No. 1, 2005, pp. 17-41.

doi:10.1007/s00766-005-0016-3

[30] C. Ghezzi and S. Guinea, "Run-Time Monitoring in Service-Oriented Architectures," *Test and Analysis of Web Services*, 2007, pp. 237-264.

[31] A. Sutcliffe, S. Fickas and M. M. Sohlberg, "Pc-re: A Method for Personal and Contextual Requirements Engineering with Some Experience," *Requirements Engineering*, Vol. 11, No. 3, 2006, pp. 157-173. doi:10.1007/s00766-006-0030-0

[32] W. L. Johnson and M. Feather, "Building an Evolution Transformation Library," *The* 12*th International Conference on Software Engineering*, IEEE Computer Society Pess, Los Alamitos, 1990, pp. 238-248.

[33] G. Coulson, G. Blair, P. Grace, F. Taiani, A. Joolia, K. Lee, J. Ueyama and T. Sivaharan, "A Generic Component Model for Building Systems Software," *ACM Transactions on Computer Systems*, Vol. 26, No. 1, 2008, pp. 1-42. doi:10.1145/1328671.1328672

[34] H. J. Goldsby, P. Sawyer, N. Bencomo, B. H. C. Cheng and D. Hughes, "Goal-Based Modeling of Dynamically Adaptive System Requirements," *The* 15*th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, IEEE Computer Society, Washington, 2008, pp. 36-45.

[35] J. Y. Halpern, "Reasoning about Uncertainty," MIT Press, Cambridge, 2003.

[36] H. Courtney, "20/20 Foresight: Crafting Strategy in an Uncertain World," Harvard Business School Press, Boston, 2001.

[37] B. H. Cheng, P. Sawyer, N. Bencomo and J. Whittle, "A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty," *The* 12*th International Conference on Model Driven Engineering Languages and Systems*, Springer-Verlag, Berlin, 2009, pp. 468-483.

[38] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng and J. Bruel, "Relax: Incorporating Uncertainty into the Specification of Self-Adaptive Systems," *Proceedings of the* 17*th IEEE International Requirements Engineering Conference*, Atlanta, 31 August-4 September 2009, pp. 79-88.

[39] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng and J. Bruel, "Relax: A Language to Address Uncertainty in Self-Adaptive Systems Requirement," *Requirements Engineering*, Vol. 15, No. 2, 2010, pp. 177-196. doi:10.1007/s00766-010-0101-0

[40] B. H. C. Cheng, *et al*., "Software Engineering for Self-Adaptive Systems: A Research Roadmap," *Software Engineering for Self-Adaptive Systems*, Springer-Verlag, Berlin, 2009, pp. 1-26.

[41] Y. Brun, G. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Muller, M. Pezze and M. Shaw, "Engineering Self-Adaptive Systems through Feedback Loops," *Software Engineering for Self-Adaptive Systems*, Springer-Verlag, Berlin, 2009, pp. 48-70.

[42] H. Muller, M. Pezze and M. Shaw, "Visibility of Control in Adaptive Systems," *The* 2*nd International Workshop on Ultra-Large-Scale Software-Intensive Systems*, ACM, New York, 2008, pp. 23-26.

[43] S. Dustdar, K. M. Goeschka, H. Truong and U. Zdun, "Self-Adaptation Techniques for Complex Service-Oriented Systems," *The* 5*th International Conference on Next Generation Web Services Practices*, Prague, 9-11 September 2009, pp. 37-43.