A Fuzzy Service Adaptation based on QoS Satisfaction

Barbara Pernici and S. Hossein Siadat

Politecnico di Milano, Italy
{pernici,siadat}@elet.polimi.it

Abstract. Quality of Service (QoS) once defined in a contract between two parties may change during the life-cycle of Service-Based Applications (SBAs). Changes could be due to system failures or evolution of quality requirements from the involved parties. Therefore, Web Services need to be able to adapt dynamically to respond to such changes. Adaptation and evolution of services are playing an important task in this domain. An essential issue to be addressed is how to efficiently select an adaptation while, there exists different strategies. We propose a fuzzy service adaptation approach that works based on the degree of QoS satisfaction. In particular, we define fuzzy parameters for the QoS property descriptions of Web Services. This way, partial satisfaction of parameters is allowed through measuring imprecise requirements. The QoS satisfaction degree is measured using membership functions provided for each parameter. Experimental results show the effectiveness of the fuzzy approach using the satisfaction degree in selecting the best adaptation strategy.

1 Introduction

In Service-Based Applications (SBAs), Quality of Service (QoS) parameters may change during the life cycle of the application. Web service adaptation is an important phase to deal with such changes. Handling changes in a demanding and adaptive environment is a vital task. One main issue lies in QoS property descriptions of Web Services. This involves specifying service requirements in a formal way, monitoring and dynamically adapting and evolving the services with respect to the QoS changes. Static adaptation is impractical due to the changing environment and high cost of maintenance and development. Specifying all possible alternative behaviour for adaptation at design time is impossible. Therefore, a declarative approach is required at run-time to support adaptation decisions.

In order to perform run-time decisions for adaptation in a volatile environment, one issue is to consider the imprecise evaluation of QoS properties. Existing approaches do not allow partial satisfaction of parameters. It is required that services should be able to tolerate a range of violation in their quality description. However, handling this toleration need to be done with special care. An important issue to address in SBAs is to what extent the QoS parameters of a

Web Service are satisfiable. The answer to this issue could be a basic for making adaptation decisions. However, this issue has not been addressed adequately in the literatures. Evaluating the extent of parameter satisfaction is necessary to help the selection of best adaptation strategy.

As an initial step to this, in [3] we provided conditions under which QoS changes are acceptable. We used a temporal logic namely Allen's Interval Algebra (AIA) [2] to formally specify the non-functional properties of web services. We then used the AIA to reason about changes of quality parameters and their evolution. In this paper, we extend [3] and propose a fuzzy approach to support service adaptation and evolution. We define *fuzzy parameters* for QoS property description of Web Services. Fuzzy parameters could be considered as fuzzy sets and measured based on their value of membership. Satisfaction degree of fuzzy parameters is measured according to their actual distance of the agreed quality ranges in the contract. The goal of this paper is to provide flexibility for service specification by applying fuzzy parameters. Using a fuzzy approach allows us to deal with reasoning on the quality violations that is approximate rather than accurate. At the end, we propose different categories of adaptation that perform based on the satisfaction degree. Experimental results show the effectiveness of using the fuzzy approach over the non-fuzzy one in making decisions for adaptation.

The remainder of the paper is structured as follows. Section 2 describes the major related work. In Section 3 we present a definition for QoS property description of services through introducing fuzzy parameters. In Section 4 we specify satisfaction functions for each parameter to measure to what extend the QoS is achieved with respect to the existing contract. We explain the decision making mechanism in Section 5 that works based on the satisfaction degree. Section 6 provides experimental result using a simulator and evaluates the effectiveness of the proposed approach. Section 7 concludes the paper and discusses our future work.

2 Related Work

Deviation of quality ranges from the existing contract may produce a system failure and bring dissatisfaction for customers. To this end, the evolution and adaptation of web services are becoming two important issues in reacting to the various changes in order to provide the agreed QoS stated in the contract. Recently, many adaptation strategies and methods have been proposed in the literature. However, most of the work in service adaptation concentrates on the technical issues and definition of mechanisms for adaptation rather than considering QoS perspective. A list of adaptation strategies for repair processes in SBAs is provided in [6] and [1]. For example, [7] proposed a service replacement approach for adaptive Web Service composition and execution, while Canfora et al. [5] presented a re-composition approach dealing QoS replanning issues at run time using late binding technique. However, none of these works consider

the consequences and potential overheads of adaptation. To this end, for example, an environment for compensation of Web Service transactions is proposed in [25]. In order to consider the overall value of a change, [15] presented an approach called *value of changed information* (VOC). Furthermore, an adaptation mechanism is proposed based on VOC in [8]. However, these works have the limitation that they do not take into account the satisfaction level of services. Making adaptation decisions and evaluating them is therefore complicated and has consequences that are often neglected. Some qualitative and quantitative techniques has been proposed, however evaluating impacts of adaptation still remains as an open challenge.

One core issue to address is the definition of a flexible description for Web Services. Formulation of service specifications/requirements has been studied in the literature. In autonomic systems and in particular web services, reasoning about such specification is a hard job due to the changing environment that affects service requirements. Although a lot of research has been conducted for functional Web Service description, only a few efforts have been done with respect to non-functional properties description of Web Service. Among syntactic and semantic WS description we refer to the work done in [30], [21] and [16] which they also provided algorithms for service selection based on such description. A major limitation of those papers and other similar ones is due to not considering the partial satisfaction of the QoS attributes. With this regards, [23] provided a semantic Policy Centered Meta-model (PCM) approach for non-functional property description of web services. A number of operators (e.g. greaterEqual, atLeast) for numeric values are defined in the model for determining tradeoffs between various requests. Therefore, the approach can support the selection of Web Services that partially satisfy user constraints. In [20] and [19], the authors extend the approach proposed in [23] by proving a solution for Web Service evaluation based on constraint satisfaction problem. The approach uses utility function to present the level of preferences for each value ranges defined in the service description. However, it does not take care of adaptation issues and controlling values at run-time. In [22], the authors discuss about fixable and non-fixable properties to deal with bounded uncertainty issue. Constraint programming is used as a solution, however, there is no evaluation of the work.

It is required to provide a framework to evaluate alternatives and quantify their impact for making decision decisions. Each alternative has different degree of satisfaction and their impact has to be evaluated in order to select the best adaptation strategy. A quantitative approach applying a probabilistic modeling is used for partial goal satisfaction in [18]. Dealing with the uncertainty issue is one major problem in order to formulate and manage service specification. Thus, recently researchers are investigating to incorporate this uncertainty into the service specification. In [14], the author provides support for reasoning about uncertainty. A goal-base approach for requirement modeling in adaptive systems is proposed in [9] which uncertainty of the environment is taken into account. Furthermore, a language named RELAX is developed for specifying requirements in adaptive systems [26, 27] in which certain requirements could be temporarily

relaxed in favor of others. In general, different temporal logics have been used for formal specification of requirement. Linear Temporal Logic (LTL) has been used in [4, 11] to formally specify requirements in a goal oriented approach. In particular, LTL is extended in [31] and named A-LTL to support adaptive program semantics by introducing an adaptation operator. [3] uses Allen's interval algebra for the formal specification of service requirement. Those approaches have limitations such that they are unable to consider environmental uncertainty and behave in a binary satisfaction manner.

Fuzzy approach [29] is an alternative to concur such limitations of aforementioned approaches. However, the fuzzy approach may not be the only alternative to deal with uncertainty. Different mathematical and frameworks are presented in the literature to address the uncertainty issue and partial satisfaction of the requirements. For example, making decisions about non-functional properties using Bayesian networks is proposed in [13] while [17] used a probabilistic method for this purpose. Applying fuzzy logic to incorporate uncertainty and making decisions has been proposed in other domains such as management, economy and many aspects of computer science, however, to the best of our knowledge there is very little of such application in adaptation of web services. As of such, [10] proposed a fuzzy approach for assigning fitness degrees to service policies in a context-aware mobile computing middleware. A trade-off analysis using fuzzy approach for addressing conflicts using imprecise requirements in proposed in [28]. With respect to partial satisfaction of requirements, [12] provided a web service selection approach using imprecise QoS constraints.

There are several different approaches towards adaptation of web services. This diversity yields from a missing consensus on the required decision making to automatically perform web service adaptation. Therefore, in this paper we propose a fuzzy adaptation approach as a possible way in providing a foundation of such a consensus which is based on the satisfaction degree of QoS parameters.

3 Fuzzy Parameters for QoS Property Description

This section is concerned with QoS property descriptions of Web Services and is devoted to demonstrate the formal definitions of quality *parameters* in a service description. The formal specification we propose has been inspired and is an extension of our previous work [3]. We extended the work by taking advantage of a fuzzy approach in which we define *fuzzy parameters*. Having introduced the fuzzy parameters it is possible to understand to what extent the quality parameters are violated/satisfied.

In order to formally define fuzzy parameters, we need to introduce some background from [3]. We define set $\mathcal{D}$ to contain the quality dimensions (such as availability, execution time, price or throughput) identified and agreed by the service provider and consumer. Each quality dimension has a domain and range; e.g., availability is a probability usually expressed as a percentage in the range 0-100% and execution time is in the domain of real numbers in the range $0..+\infty$. A quality dimension d can be considered *monotonic* (denoted by d^+) or

antitonic (d^-); monotonicity indicates that values closer to the upper bound of the range are considered better, whilst with antitonic dimensions values closer to the lower bound are considered better. A *parameter* m associates a quality dimension to a value range.

If a parameter is non-fuzzy (strict) its satisfaction degree will be evaluated in a binary manner (Yes or No). In contrast, fuzzy parameters (relaxed) will be evaluated in a fuzzy manner which shows different degree of satisfaction ($x \in [0, 1]$). Note that we also provide value ranges for both parameters regardless of being fuzzy or non-fuzzy. The satisfaction degree of parameters will be evaluated using *membership functions* we introduce in the next section. In the following we provide the extended definition of a *parameter* based on the definition introduced in [3].

Definition 1 (Parameter). *We define a Parameter* $m \in \mathcal{M}$ *as a tuple* $m := (d, v, f), d \in \mathcal{D}, v \in \mathcal{V}, f \in \{s, r\}$. *where* $\mathcal{D}$ *is the set of quality dimensions,* $\mathcal{V}$ *is the set of ranges for all quality dimensions* $\mathcal{D}$*, s represent a strict parameter and r represent a relaxed parameter.*

During a service life-cycle, QoS offerings may change due to several reasons. Therefore, adaptation of web services needs to be performed in an appropriate manner to accommodate QoS changes/violations by choosing the best adaptation strategy. Defining service description with the aforementioned fuzzy parameters provides a more flexible situation dealing with adaptation decisions. We discuss how it can facilitate the adaptation of web services through an example. According to the new definition of parameters, we consider availability and response time as fuzzy parameters. Let us assume a contract with a initial value range of availability between 80% to 90% and response time between 2 to 5 seconds.

In [3] we provided situations in which new QoS ranges could be still acceptable for both parties according to the existing contract. We defined a compatibility mechanism that uses parameter subtyping and used Allen's Interval Algebra [2] in order to express the subtyping. The provider and requestor are compatible with each other according to the existing contract if the QoS changes are in one of the acceptable situations. If the compatibility is not provided, however it does not give any information about the degree of satisfaction/dissatisfaction of the offered service. For example if the new range of availability is less than 80%, this is not considered as an acceptable situation and it is considered as a violation. In such cases, we would also like to understand to what extent the quality parameter and the aggregated service quality are satisfactory. An availability of 75% might still be acceptable if we consider the partial satisfaction of quality ranges.

4 Specifying Satisfaction Function

Having defined the fuzzy parameters we are able to apply the fuzzy logic. As for the first step we need to know the right amount of quality satisfaction.

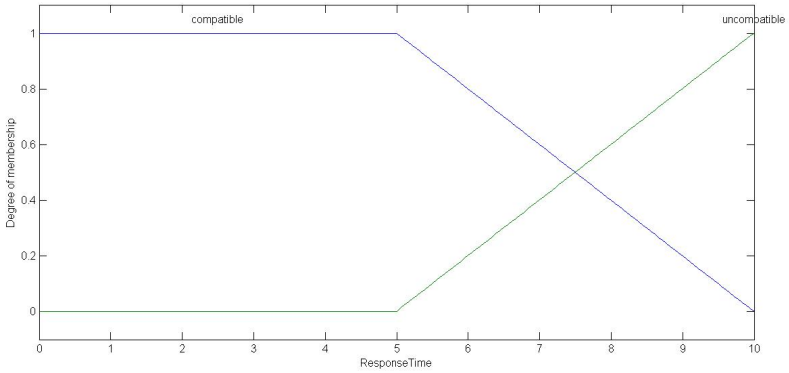


Fig. 1: Membership functions for response time

Previously in [3], we provided a compatibility mechanism to understand under which conditions the changes are acceptable. The approach suffers from the limitation that changes are considered either compatible or incompatible with the contract. This means, quality changes are calculated in a binary approach which it does not take into account clearly the relation of quality parameters with their satisfaction. To say it in other way, the QoS parameters are measured in a precise manner and their partial satisfaction is not taken into account. In the following we provide mechanisms to allow partial satisfaction of quality parameters imprecisely using fuzzy sets.

The main point of using fuzzy logic is to find a relation and to map our input space to the output space. The inputs here are namely service availability and response time and the output is the overall satisfaction degree of them. For each QoS parameter in the service description we provide a membership function that represent the level of satisfaction of each parameter. The membership functions map the value of each parameter to a membership value between 0 and 1. We use a piece-wise linear function, named *trapezoidal* membership function, for this purpose. Membership functions for ResponseTime and availability are shown in figures 1 and 2.

Having defined the membership functions, the mapping between the input and output space will be done by defining a list of *if-then statements* called *rules*. We have already defined what do we mean being compatible and incompatible for the quality parameters and specified their ranges using membership functions. Since we are relaxing the antecedent using a fuzzy statement, it is also required to represent the membership degree of the output (i.e. here satisfaction). Therefore, the satisfaction degree is also represented as fuzzy sets: satisfaction is low, satisfaction is average and satisfaction is high.

We define three if-then rules as below. it represents the *antecedent* and *consequent* of the rule. All the rules are applied in parallel and their order in unimportant. We define the fuzzy union/disjunction (OR) and the fuzzy conjunction/intersection (AND) using *max* and *min* functions respectively. Therefore

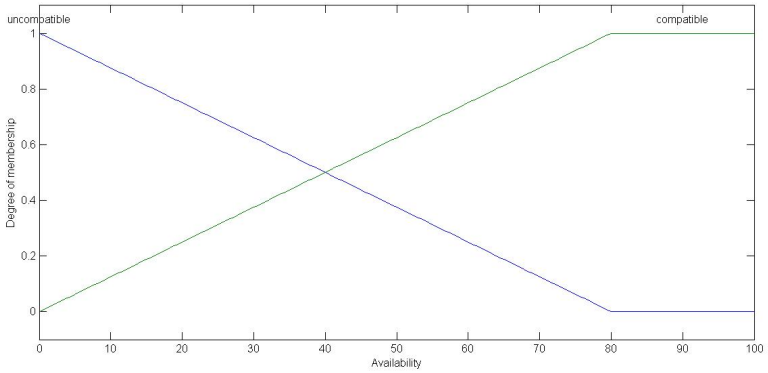


Fig. 2: Membership functions for ResponseTime and Availability

$A\,AND\,B$ is represented as $min(A, B)$ and $A\,OR\,B$ is represented as $max(A, B)$.

1. If $(ResponseTime\,is\,compatible)$ and $(Availability\,is\,Compatible)$ $then$ $(Satisfaction\,is\,high)$.
2. If $(ResponseTime\,is\,incompatible)$ or $(Availability\,is\,incompatible)$ $then$ $(Satisfaction\,is\,average)$.
3. If $(ResponseTime\,is\,incompatible)$ and $(Availability\,is\,incompatible)$ $then$ $(Satisfaction\,is\,low)$.

5 Decision Making for Adaptation and Evolution

We use the satisfaction degree calculated using the fuzzy inference system for the adaptation and evolution decision making. The decision making mechanism works based on the algorithm we provided in [24]. The algorithm evaluates the evolution of the service and decides which adaptation strategy to take with respect to the predefined threshold degree for QoS satisfaction. The two main decisions are the *internal renegotiation* in which the changes are compatible with the service description in the contract and *service replacement* in which the changes are incompatible with the existing contract. The former case deals with the internal contract modification with the same provider and requester while the earlier case requires the selection of a new service and establishment of a new contract which can result in a huge loss of time and money.

Having provided such a decision making mechanism allows us to offer a flexible adaptation mechanism. This is done by identifying threshold to what constitutes *compatible* and *incompatible*. Using satisfaction degree allows us to define the criticality of a change/violation. Therefore, we are able to understand whether a violation is critical and it results in a service replacement or the violation is still acceptable. This way, a slight change from the quality ranges defined in the contract will not trigger the adaptation. Table 1 shows the result of check-

ing for compatibility for a possible set of changes. The comparison is between our fuzzy approach and a traditional non-fuzzy one that works based on the precise evaluation of the quality ranges in the contract.

Number	Change	Replacement? (Non-fuzzy/Fuzzy)	Change	Replacement? (Non-fuzzy/Fuzzy)
1	$S_1 = (6, .90)$	Yes/No	$S_2 = (7, .75)$	Yes/Yes
2	$S_3 = (5, .85)$	No/No	$S_4 = (3, .70)$	Yes/No
3	$S_5 = (2, .85)$	No/No	$S_6 = (2, .78)$	Yes/No
4	$S_7 = (2, .60)$	Yes/Yes	$S_8 = (6, .90)$	Yes/No
5	$S_9 = (7, .95)$	Yes/Yes	$S_{10} = (6, .50)$	Yes/Yes

Table 1: Comparing the adaptation decisions using fuzzy and non-fuzzy approach

For example in $S_1 = (6, .90)$, changing the response-time to 6s will not result a service replacement applying the fuzzy approach since it has the satisfaction degree of almost 83%. While applying a non-fuzzy approach, it is considered a violation because it does not respect the initial response-time range (2, 5) in the contract. However, if a change results in a low satisfaction degree, service replacement is necessary in both approaches as in the case $S_{10} = (6, .50)$ which the satisfaction degree is around 62%.

6 Experiments and Implementation

Having defined the membership functions and rules in the previous sections, we have built and simulated a fuzzy inference system to interpret rules. The process has different steps including: fuzzification of input quality parameters, applying fuzzy operators to the antecedent, implication from the antecedent to the consequent, aggregation of the results for each rule, and defuzzification. A view of the simulator including the previous steps is illustrated in figure 3 in which a complete fuzzy inference system is represented.

The first step is to apply the membership functions to map each QoS parameters to the appropriate fuzzy set (between 0 and 1). We used two inputs of Availability (interval between 0 to 100) and Response-time (interval between 0 to 10). The inputs are mapped to fuzzy linguistic sets: availability is compatible, availability is incompatible, response-time is compatible, and response-time is incompatible. Figures 1 and 2 show to what extent the availability and response-time are compatible. The next step is to give the result of the fuzzified input parameters to the fuzzy operators. According to the rules, AND and OR operators are applicable. This will give us a degree of support for each rule. Next is applying the implication method that uses the degree of support to calculate the output fuzzy set. We used a *minimum* method to truncate the output fuzzy set for all the rules separately. However, we apply all the rules in parallel and we do not define any priority and weight for them.

At the end of the implication, we apply an aggregation method to combine all the rules. This way, the outputs of each rule represented in fuzzy sets are combined into a single fuzzy set. A maximum method is used for the aggregation.

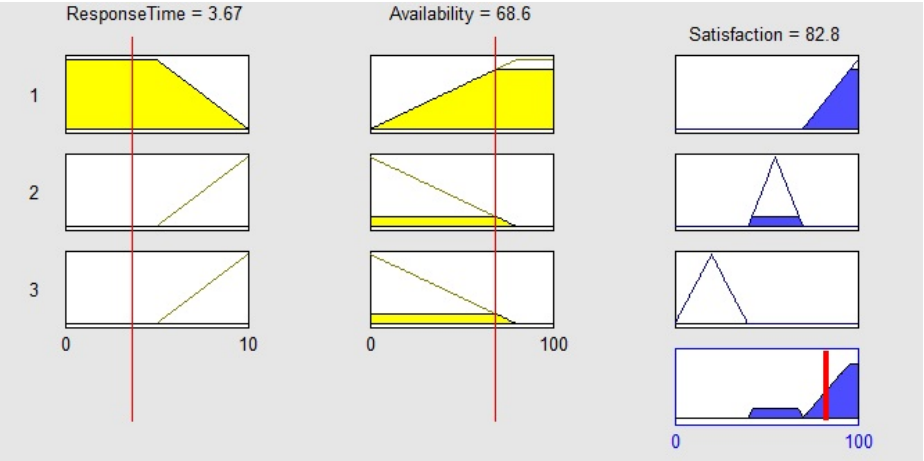


Fig. 3: A view of the simulator for fuzzy inference system

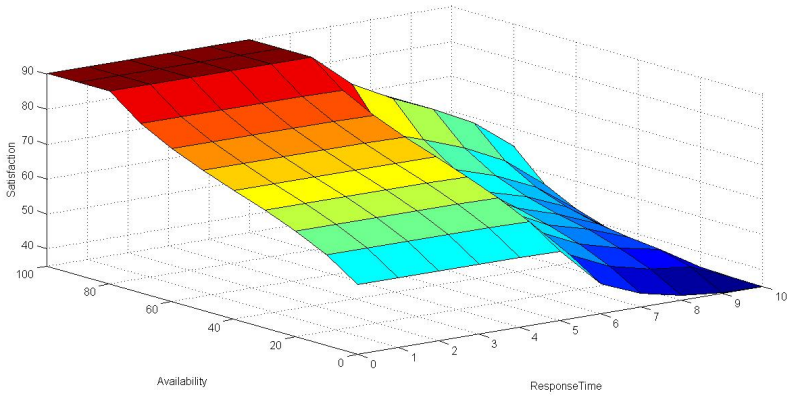


Fig. 4: The output of satisfaction degree according to ResponseTime and Availability membership function

The last step is to defuzzify the fuzzy set resulted after the aggregation step. We applied a *centroid* method to calculate the defuzzification process. The method returns the center of the area under the curve. Figure 3 shows that the response-time of 3.67 seconds and availability of 68.6% result a satisfaction degree of 82.8. Figure 4 shows a surface map for the system and the dependency of the satisfaction degree on the response-time and availability.

We evaluate the effectiveness of the fuzzy approach with a non-fuzzy approach with respect to the stability of the system in terms of number of times a service needs to be replaced. The fuzzy approach performs the adaptation based on the QoS satisfaction. Only if the result of the satisfaction is lower than a threshold a service replacement occurs. While in the non-fuzzy approach, the replacement decision is done based on the precise evaluation of the QoS value ranges.

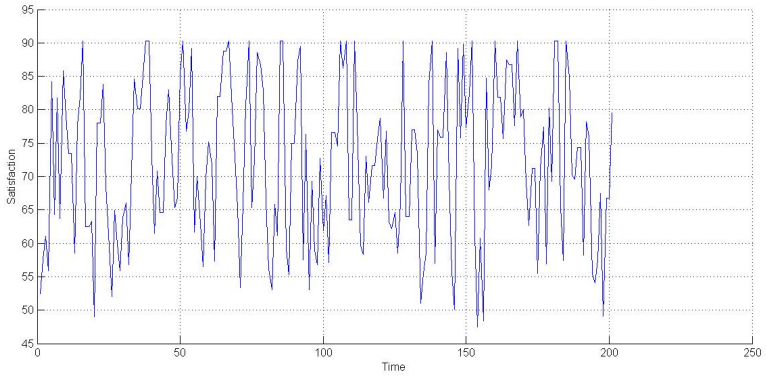


Fig. 5: Satisfaction degree

We have conducted our experiment 200 times, each time providing random data for the input parameters. Figure 5 illustrates the output (satisfaction degree) of the experiment. The satisfaction threshold was set to 70%.

Figure 6 represents the stability of the fuzzy and non-fuzzy systems. As it is shown, the number of service replacement in a non-fuzzy approach is much higher than when we apply a fuzzy approach. This actually is a direct proof of our approach. Using fuzzy parameters we allow partial satisfaction of the parameters. Therefore, the decision making for adaptation is not based on the precise evaluation of the quality ranges and it is rather imprecise and allows the parameters to be relaxed. The non-fuzzy approach involved the maximum number of service replacement which includes more queries for the service selection. This can results in a huge loss of time and money. The cost of establishing a new contract is also considerable.

7 Conclusions and Future Work

In this paper, we used fuzzy parameters for the QoS property descriptions of Web Services and a fuzzy approach is taken in order to select adaptation strategy. However, interpreting and presenting adaptation decisions based on fuzzy logic is still a hot research area that requires to be investigated more in the research community of software and service engineering.

In particular, we used linear *trapezoidal* membership function for the sake of simplicity. Currently, we are conducting more experiment to investigate the usage of *Gaussian* distribution function and *Sigmoid* curve that have the advantages of being smooth and non-zero all the time.

As for the future work, we aim to continue exploring the use of fuzzy parameters for the QoS matching. Applying more sophisticated functions using AI to Map the satisfaction degree to the appropriate adaptation decision might be worth exploring. However, there are still challenges that need to be addressed.

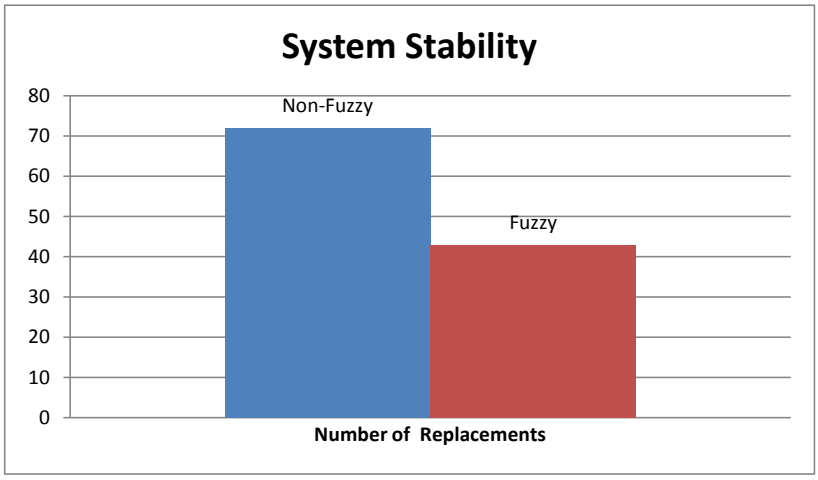


Fig. 6: System stability of using fuzzy and non-fuzzy approach

For example, to what extent a parameter could be relaxed yet consider no violation? We also plan to incorporate more QoS parameters for calculating the overall satisfaction degree that influence the process of decision making. This requires the definition of more complex rules that represent the relation and dependencies between parameters.

Last but not least, applying an appropriate decision making requires an analytical evaluation based on a cost model. We would like to know under which circumstances the proposed approach is beneficial considering both QoS and business value criteria.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

1. E. Di Nitto R. Kazhamiakin V. Mazza A. Bucchiarone, C. Cappiello and M. Pistore. Design for adaptation of service-based applications: Main issues and requirements. In *the Fifth International Workshop on Engineering Service-Oriented Applications: Supporting Software Service Development Lifecycles (WESOA)*, 2009.

2. J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.
3. Vasilios Andrikopoulos, Mariagrazia Fugini, Mike P. Papazoglou, Michael Parkin, Barbara Pernici, and Seyed Hossein Siadat. Qos contract formation and evolution. In *EC-Web*, pages 119–130, 2010.
4. Greg Brown, Betty H. C. Cheng, Heather Goldsby, and Ji Zhang. Goal-oriented specification of adaptation requirements engineering in adaptive systems. In *SEAMS '06: Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pages 23–29, New York, NY, USA, 2006. ACM.
5. Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. Qos-aware replanning of composite web services. In *ICWS*, pages 121–129, 2005.
6. Cinzia Cappiello and Barbara Pernici. Quality-aware design of repairable processes. In *the 13th International Conference on Information Quality (ICIQ 08)*, pages 382–396, 2008.
7. Girish Chafle, Koustuv Dasgupta, Arun Kumar, Sumit Mittal, and Biplav Srivastava. Adaptation in web service composition and execution. In *ICWS*, pages 549–557, 2006.
8. Girish Chafle, Prashant Doshi, John Harney, Sumit Mittal, and Biplav Srivastava. Improved adaptation of web service compositions using value of changed information. In *ICWS*, pages 784–791, 2007.
9. Betty H. Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, MODELS '09, pages 468–483, Berlin, Heidelberg, 2009. Springer-Verlag.
10. Ronnie Cheung, Jiannong Cao, Gang Yao, and Alvin T. S. Chan. A fuzzy-based service adaptation middleware for context-aware computing. In *EUC*, pages 580–590, 2006.
11. Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20:3–50, April 1993.
12. Martine De Cock, Sam Chung, and Omar Hafeez. Selection of web services with imprecise QoS constraints. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, WI '07, pages 535–541, Washington, DC, USA, 2007. IEEE Computer Society.
13. N. Fenton and M. Neil. Making decisions: using bayesian nets and mcda. *Knowledge-Based Systems*, 14(7):307 – 325, 2001.
14. Joseph Y. Halpern. *Reasoning about Uncertainty*. MIT Press, Cambridge, MA, USA, 2003.
15. John Harney and Prashant Doshi. Adaptive web processes using value of changed information. In *International Conference on Service-Oriented Computing (ICSOC*, pages 179–190, 2006.
16. Kyriakos Kritikos and Dimitris Plexousakis. Semantic QoS-based web service discovery algorithms. In *ECOWS*, pages 181–190, 2007.
17. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Probabilistic symbolic model checking with prism: A hybrid approach. In *TACAS*, pages 52–66, 2002.
18. Emmanuel Letier and Axel van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. *SIGSOFT Softw. Eng. Notes*, 29:53–62, October 2004.

19. Pei Li, Marco Comerio, Andrea Maurino, and Flavio De Paoli. Advanced non-functional property evaluation of web services. In *Proceedings of the 2009 Seventh IEEE European Conference on Web Services*, ECOWS '09, pages 27–36, Washington, DC, USA, 2009. IEEE Computer Society.
20. Pei Li, Marco Comerio, Andrea Maurino, and Flavio De Paoli. An approach to non-functional property evaluation of web services. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 1004–1005, Washington, DC, USA, 2009. IEEE Computer Society.
21. Octavio Martín-Díaz, Antonio Ruiz Cortés, David Benavides, Amador Durán, and Miguel Toro. A quality-aware approach to web services procurement. In *TES*, pages 42–53, 2003.
22. Octavio Martín-Díaz, Antonio Ruiz Cortés, José María García, and Miguel Toro. Dealing with fixable and non-fixable properties in service matchmaking. In *ICSOC/ServiceWave Workshops*, pages 228–237, 2009.
23. Flavio De Paoli, Matteo Palmonari, Marco Comerio, and Andrea Maurino. A meta-model for non-functional property descriptions of web services. In *Proceedings of the 2008 IEEE International Conference on Web Services*, pages 393–400, Washington, DC, USA, 2008. IEEE Computer Society.
24. Barbara Pernici and S. Hossein Siadat. Adaptation of web services based on QoS satisfaction. In *WESOA '10: Proceedings of the 6th International Workshop on Engineering Service-Oriented Applications*, Berlin, Heidelberg, 2010. Springer-Verlag.
25. Michael Schäfer, Peter Dolog, and Wolfgang Nejdl. An environment for flexible advanced compensations of web service transactions. *ACM Trans. Web*, 2:14:1–14:36, May 2008.
26. Jon Whittle, Peter Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean-Michel Bruel. Relax: Incorporating uncertainty into the specification of self-adaptive systems. In *RE*, pages 79–88, 2009.
27. Jon Whittle, Peter Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean-Michel Bruel. Relax: a language to address uncertainty in self-adaptive systems requirement. *Requir. Eng.*, 15(2):177–196, 2010.
28. John Yen and W. Amos Tiao. A systematic tradeoff analysis for conflicting imprecise requirements. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, RE '97, pages 87–96, Washington, DC, USA, 1997. IEEE Computer Society.
29. Lotfali Askar Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
30. Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.
31. Ji Zhang and Betty H. C. Cheng. Using temporal logic to specify adaptive program semantics. *Journal of Systems and Software*, 79(10):1361–1369, 2006.

Selection of Service Adaptation Strategies Based on Fuzzy Logic

Barbara Pernici and S. Hossein Siadat
Politecnico di Milano
Dipartimento di Elettronica e Informazione
p.zza Leonardo Da Vinci, 32 - 20133, Milano, Italy
pernici@elet.polimi.it

***Abstract*—Web Service adaptation and evolution is receiving huge interest in service oriented architecture community due to dynamic and volatile web service environment. Regarding quality of service changes, Web Services need to be able to adapt dynamically to respond to such changes. However, formulating quality of service parameters and their relationship with adaptation behaviour of a service based system is a difficult task. In this paper, a Fuzzy Inference System (FIS) is adopted for capturing overall QoS and selecting adaptation strategies using fuzzy rules. The overall QoS is inferred by QoS parameters, while selection of adaptation strategies is inferred by the overall QoS, importance of QoS and cost of service substitution. In particular, hierarchical fuzzy systems were used to reduce the number of rules. Our approach is able to efficiently select adaptation strategies with respect to QoS changes. We test and compare our fuzzy inference adaptation with a naive adaptation approach that works based on precise measurement of QoS in order to show the performance of the approach in reducing the number of service substitutions for minor deviations.**

***Keywords*-QoS; service adaptation and evolution; fuzzy logic; service based applications;**

I. Introduction

The performance of Service Based Applications (SBAs) [1] is relied upon providing a Service Level Agreement (SLA). SLA is a mutually agreed contract between a service provider and consumer that describes non-functional properties of web services. QoS parameters are often changing due to dynamic and volatile service environment. In such environment, Web Services need to be able to adapt dynamically trying to respect the SLA. For this, changes are required to be captured, evaluated and proper actions need to be taken accordingly. Therefore, adaptation and evolution of Web services are of great importance in order to guarantee the quality of service (QoS) defined in a contract. With this regard, we intend to address the following key issues in our work.

- **Selecting Service Adaptation Strategies.** Depending on the adaptation triggers, there exist a set of adaptation strategies. However, selection of the most suitable adaptation strategies is a complex task due to the multiple criteria involved in the process on decision making [2]. For example different recovery/adaptation actions have different time and computation complexity. Impact and scope of change should also be considered as adaptation requirements. Moreover, adaptation cost needs to be taken into account for economical reason. Therefore, understanding the adaptation triggers that may produce a faulty situation and adaptation requirements is necessary to make an appropriate decision for adaptation and it has to be done with a special care. We argue what is missing here is an appropriate decision making process for selecting adaptation strategies that takes into consideration different adaptation requirements. In this paper we are interested in changes of service quality as symptoms and triggers for adaptation. Besides, we consider cost of service substitution and importance of QoS as requirements/factors of adaptation for selecting the best strategy.
- **Quality of Service Management.** A SBA has some very nonlinear characteristics from the QoS perspective. However, current approaches mostly use precise methods for QoS property description and quality measurement. This means that QoS parameters are defined using crisp values and their evaluation is based on precise measurement of defined ranges in the contract. Therefore, any deviation from quality ranges is considered a violation, albeit minor, and thus an adaptation action (mostly service substitution) takes place regardless to its consequences. Moreover, partial satisfaction of QoS parameters can not be supported by applying precise methods. However, applying precise methods are not suitable for all circumstances due to uncertainty issues. For example, large number of QoS parameters and unclear relation between parameters makes it difficult and complex to formulate an analytical optimization model for QoS management in SBAs.

Soft computing is an alternative when above problems occur by providing inexact solutions for computationally-hard tasks and relaxing parameters. Applying a soft computing approach is beneficial for defining QoS description and measuring quality parameters in order to compute the overall QoS. *Inference* methods are used when the input-output relation can be expressed in the form of *if-then rules*. Fuzzy logic [3] is used in this category and describe the system behaviour by using simple rules. In general, fuzzy logic is particularly suitable among other techniques in dealing with

uncertainty and using imprecise parameters. The definition of quality parameters such as dependability, security and reputation has a high degree of unclarity and ambiguity that makes it difficult to define them in a precise approach using crisp values. Furthermore, mapping such quality parameters to the overall QoS is difficult to be defined using mathematical expression. In this context, *linguistic variables* [4] can be used to define QoS parameters. As an example, reputation can be expressed with three states of *low, medium* and *high*.

In this paper, we take advantage of the fuzzy logic for measuring overall QoS and selecting service adaptation strategies. We concentrate on QoS changes in a SBA and their relations with adaptation actions. We apply *hierarchical fuzzy systems* to address the rule explosion problem and to reduce the number of rules. This provides the ability to efficiently manage different categories of input parameters, increase the scalability and improve the reusability of the system. Our approach employs two fuzzy inference systems that use if-then rules to manage QoS changes and to support a decision making to decide which adaptation strategy is the best to be taken. We consider three adaptation strategies: do-nothing, renegotiation, and substitution.

The paper is organized as follows. Section II describes the major related work. The approach overview and the extended architecture and life cycle of a SBA associated with fuzzy inference engines are described in Section III. Section IV introduces the main basic of fuzzy logic systems and explains the fuzzy inference mechanism. Rules are defined and QoS changes are evaluated using a fuzzy inference methodology. Section V shows the experiments in applying a fuzzy inference system for fault detection and adaptation of services through our working example. Section VI concludes the paper and discusses our future work.

II. Related Work

Recently, much work has been done in the area of adaptation of service base application. Among those we can refer to works by [5] and [6]. Furthermore, cross-layer adaptation of service based application is also investigated in literature [7], considering various factors from different SBA layers that influence the adaptation behaviour of the system. However, most of the work takes into account the functional perspective of services and less attention has been paid to non-functional aspects. QoS contracting issues is investigated in [8] and [9]. There are only few approaches that provide support for QoS adaptation of SBAs. [10] proposed an approach for QoS-aware adaptation based on dynamic replanning of service composition. Adaptation of SBAs based on process quality factor analysis is addressed in [11]. However, our approach is different in that we consider the partial satisfaction of QoS parameters through imprecise QoS measurement, which is not considered in the above approaches due to the problem of QoS property description we already mentioned in the Section I.

QoS property description is studied in the literature [12] in which QoS parameters are measured through mathematical formulas. The measurement for each quality parameter depends on the value of the same parameter from other services in the composition. For example it requires the maximum and minimum value of each parameter for scaling phase. However, formulating quality of service parameters and their relationship with adaptation behaviour of a SBA is a difficult task. The work in [13], proposed an approach that allows partial satisfaction of QoS attributes by exploiting hierarchical constraint logic programming. However, the overall QoS is not measured and its relation with quality parameters are not considered.

The work in [14] discusses the impact analysis of repair actions from a business perspective. The authors present mathematical models to compute economic values of applying different repair actions based on satisfaction index of business entities and cost of adaptation. While similar in spirit, we consider changes of QoS parameters as triggers for adaptation and a trade-off approach between quality, its importance and cost is taken for the analysis which is based on fuzzy inference rather than applying mathematical models.

Fuzzy logic has been applied to many fields (e.g. control and communication) for solving problems and decision making where the relation of existing parameters are too complicated to be modelled by conventional mathematical models and techniques. With respect to QoS, the definition of quality parameters is associated with uncertainty due to the complex and dynamic environment. In this context, fuzzy logic seems to be a promising approach for QoS management. [15] presented an approach using fuzzy control for improving QoS performance. Other approaches address the problem of QoS management in distributed multimedia applications such as those in [16] and [17]. However, these systems are either problem-specific or application specific (e.g. Multimedia applications). Moreover, they are very dependent to the specific scenario of the application.

Recently, researchers have demonstrated increasing attention towards fuzzy QoS adaptation. Soft computing techniques, in particular fuzzy logic, for QoS adaptation are addressed in [18]. An approach for adaptive middleware infrastructure is presented in [19] for context-aware mobile applications. The approach defines policies and calculates their fitness degree using fitness functions which will be compared with the current fuzzified context situations to infer adaptation decisions. Our idea in applying hierarchical fuzzy inference is closely related to the work in [20]. The adaptation decisions are taken by inferring the user expectation of QoS and network quality. Our approach follows a broader range, since we consider other adaptation requirements (e.g. cost aspects) for inferring adaptation decisions.

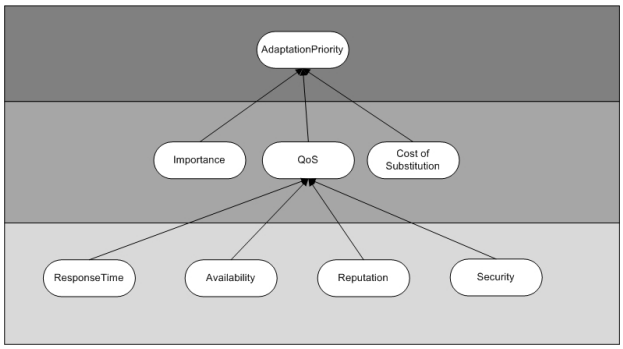


Figure 1. Hierarchy approach: the quality model and adaptation requirements

III. Approach Overview

Our proposed approach for selecting adaptation strategies is mainly based on quality changes as adaptation triggers. For this purpose, we use a quality model that is illustrated in Figure 1. Other adaptation requirements, i.e. cost of adaptation and importance of QoS, are also shown in the middle level of the figure. We consider four quality parameters in our web service quality model: response time, availability, security and reputation. In principle, our quality model can be extended to include quality metrics such as those proposed in [21], [22] and [12]. The approach uses a hierarchy fuzzy system [23], [24]. Particularly, we use two fuzzy inference engines, namely, *QoS assessment engine* and *decision making engine*. The former is used to infer the overall degree of QoS and the earlier is used to choose the adaptation actions.

Each inference engine uses its own fuzzy rules. QoS-related rules are stored in a *knowledge base* and adaptation-related rules are stored in a *adaptation rule base*. QoS parameters perform as input variables for the QoS assessment engine. Fuzzy rules in the knowledge base can be used for inferring the overall QoS degree. The decision making engine works based on the results of the QoS assessment engine. Fuzzy rules in the adaptation rule base will be used for inferring an adaptation strategy. The fuzzification process and steps of a fuzzy inference systems will be described in the next section.

It is also possible to include other types of parameters for example context-aware, CPU and other resource parameters that may influence the overall QoS. In this case we may use a separate fuzzy inference engine for a specific category of parameters. Other parameters are not yet included in our approach. However, it can be done by a *change analyser* to distinguish different category of parameters and send them to their specific fuzzy inferring engines. Note that in this paper we only use QoS parameters in our quality model.

Increasing the number of QoS parameters can exponentially increase the number of rules in a FIS system which eventually increase computational time. Applying the hierarchical fuzzy systems provides scalability to the system. This way can lead to a notable decrease in the number of rules. Furthermore, it increases the re-usability of the system and provides support for updating the rules independently. For example, if we need to involve a new parameter for calculating the overall QoS, only the QoS-related rule repository will be updated. In general, using the hierarchical fuzzy rules allows the system to define independent fuzzy rules for each category. Therefore, changes in input parameters of one category do not affect rules in the other category and only corresponding fuzzy rules will be modified.

A high-level overview of our approach is illustrated in Figure 2. The approach follows the basic architecture and life cycle of a SBA and includes phases of *contract formation*, *execution*, *monitoring* and *adaptation*. The focus of the model is on the monitoring and adaptation phases.

Fuzzy QoS Assessment. A fuzzy QoS assessment component is placed in the monitoring phase. It includes the QoS assessment engine and its corresponding QoS rules stored in the knowledge base. QoS parameters captured by QoS monitors are used as inputs for the QoS assessment engine. Therefore, the engine takes the QoS parameters as inputs and apply the QoS rules from the knowledge database to provide the overall degree of the QoS. Overall, the quality parameters will be monitored and evaluated using the QoS assessment engine.

Decision Making. A decision making engine is placed in the adaptation phase for selecting adaptation strategies. The engine uses the overall degree of QoS received from the QoS assessment engine together with other adaptation factors. Information about adaptation requirements are kept in the adaptation rule base. The output of the engine, after defuzzification, represent the adaptation strategy needs to be taken.

IV. Fuzzy Logic and Fuzzy Inference Systems

The main idea of using fuzzy logic is to understand the relation between input and output parameters and to map the input space to the output space. In the following we describe how a fuzzy inference system works.

A fuzzy inference system consists of five major steps. The first step is *fuzzification* of input parameters. Crisp parameters are converted to linguistic fuzzy parameters using membership functions. We use quality parameters from our quality model in Figure 1. Measurable parameters such as availability, response-time and reputation (to some extent) can be measured using mathematical techniques such as the one in [12]. The crisp numerical values can be taken from service level agreements which define specific ranges for each parameter. For non-measurable parameter, such as security, approaches for converting value to level can be used [8]. Each input parameter then should be defined by a set of fuzzy linguistic. We use three states of *low, medium* and *high* for all the quality parameters for the sake of simplicity,

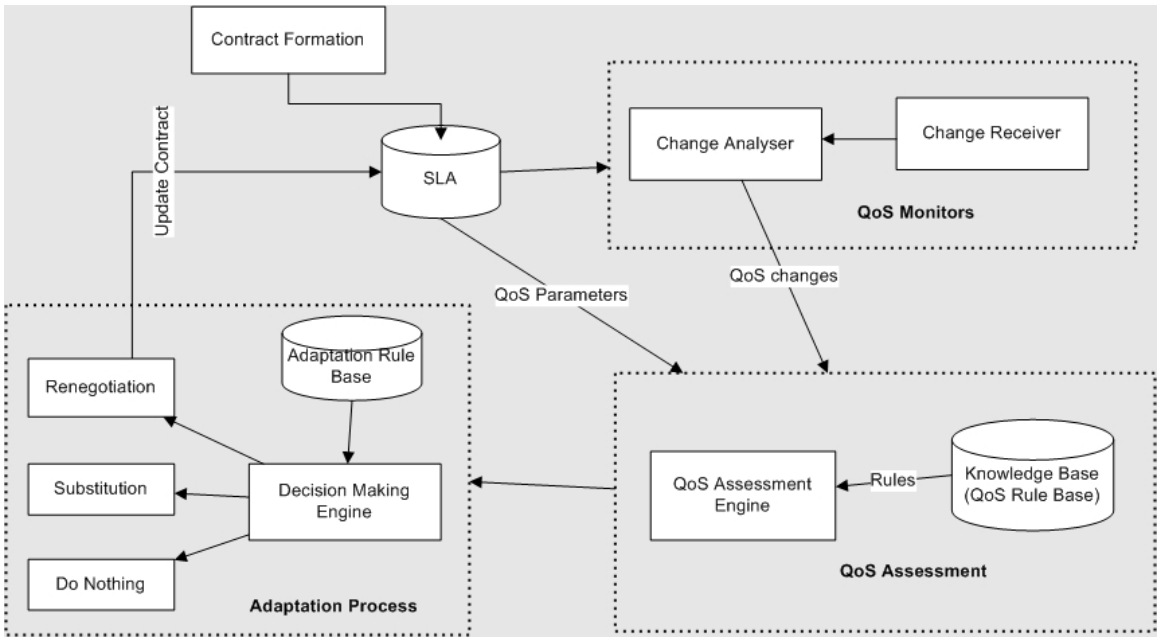


Figure 2. Approach Overview

while the output quality parameter, overall QoS, is presented by five states of *very-low, low, medium, high* and *very-high*.

Then we need to know the mathematical meaning of the linguistic variable. What does it mean to say *availability is high* and to what extent?. Therefore, a membership function shows the degree of affiliation of each parameter by mapping their values to a membership value between 0 and 1. Different membership functions could be used for such mapping in which basic functions are namely *piece-wise linear*, *the Gaussian distribution*, *the sigmoid curve*. The linear functions have the advantage of simplicity while the Gaussian functions take the advantage of smoothness and being non-zero all the time.

For measurable parameters, the primary range of each parameter is taken from the contract and interpreted to different states. For example the response-time of $[0..3]$ is good, $[3..7]$ is medium and $[7..10]$ is weak. As for the non-measurable parameter, different levels is converted to the corresponding state. For example, A level security is high, B level is medium and C level is low. The membership function of reputation parameter is illustrated in Figure 3 while the membership function of service quality is shown in Figure 4. Gaussian Membership functions are used for all the input and outputs variables in this paper.

All the inputs must be fuzzified before applying fuzzy rules. Given the fuzzy inputs and membership functions, fuzzy *if-then* rule statements constitute the fuzzy logic of the system. The structure of a fuzzy rule with respect to multiple QoS parameters is shown in the form of equation 1 where A, B, C and D are linguistic values. The inputs are

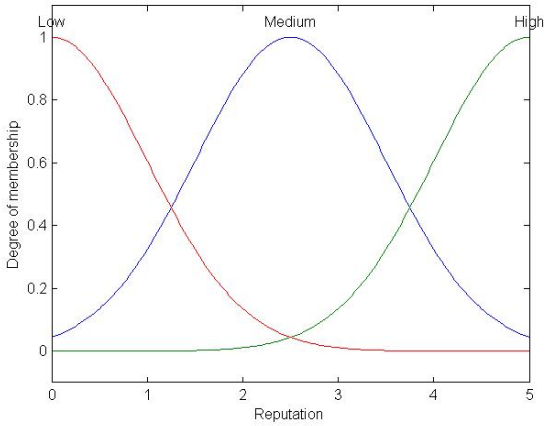


Figure 3. The membership function of reputation parameter

quality parameters P_i and the output is the overall QoS q. The if-part of the rule is called *antecedent* while the then-part is called *consequent*.

$$R: If\ P_1\ is\ A\ and\ P_2\ is\ B\ or\ P_3\ is\ C\ then\ q\ is\ D \quad (1)$$

Applying rules consists of three steps. First is evaluating the if-part by applying fuzzy operators to the antecedent. After evaluating the antecedent, the result will be applied to the consequent, which is called *implication*. The result of implication is a fuzzy set that is truncated using an implication method. Since there are more than one rule, the output fuzzy set of each rule is required to be aggregated to

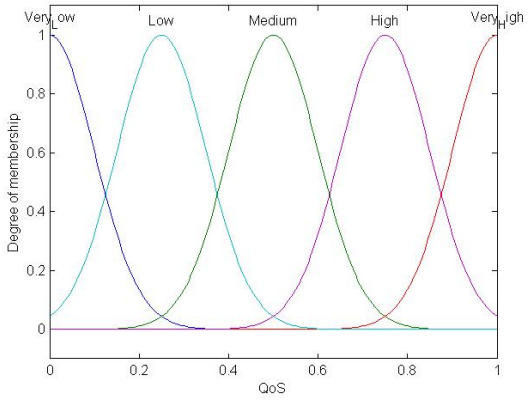


Figure 4. The membership function of Service Quality

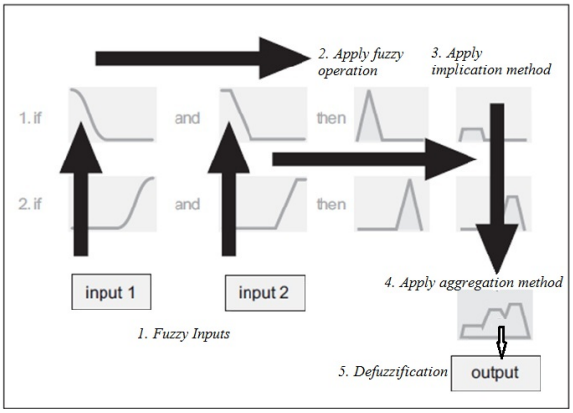


Figure 5. Steps of a fuzzy inference system with 2 inputs and 2 if-then rules

a single fuzzy set, which is called *aggregation*.

The last step is *defuzzification* that converts the fuzzy set resulted from the aggregation to a single crisp value. There are several defuzzification methods: centroid, middle of maximum, largest of maximum and smallest of maximum. Steps of a fuzzy inference system are shown in Figure 5.

V. Implementation and Results

We have implemented and tested our proposed approach for selecting adaptation strategies in a simulator. For our experiment, fuzzy inference systems are applied and the Mamdani approach is taken for generating fuzzy rules in the form of linguistic expressions. As we discussed earlier, we used particularly two fuzzy inference engines. One for inferring the overall QoS and the other one for inferring the selection of adaptation strategies called *totalQoS* and *adaptationPriority* respectively. In our experiments, We use a *min* function for the implication method, *max* function for the aggregation method and *centroid* function for the defuzzification methods. The two fuzzy inference systems are illustrated in Figure 6.

The inputs are the QoS parameters injected into the system using random values. The values are then scaled to accommodate the fuzzy logic inputs. Therefore, a saturation block is placed to keep the input data in a specified range according to the given parameter.

Each input and output value is accomplished with a membership function. The totalQoS engine uses four parameters of *response time*, *availability*, *security* and *reputation* as inputs and overall QoS as output. The overall quality of the system is given to the adaptationPriority engine as an input. *cost of substitution* and *importance* of QoS are other inputs. The adaptation engine has one output, called *actionPriority*, that represents the priority of adaptation strategies. We associated the low degree of actionPriority to *do-nothing* strategy, medium degree to *renegotiation* strategy and high degree to *substitution* strategy.

We have defined fuzzy rules using the linguistic variables for both fuzzy inference engines. We considered all possible combination of rules, 81 quality rules and 45 adaptation rules, since we conducted experiment with low number of inputs and the computational time was not a distinguishing factor in this study. However, it is possible to summarize different combinations. Fuzzy *conjunction* (AND), interpreted as a *min* function, is applied for all the rules. Figure 7 shows the QoS rules of the system. As an example, one rule is marked in the figure: when the response-time is good, availability is medium, reputation is high and security is low, then the overall QoS will be medium.

We have also defined the adaptation rules for the adaptation inference engine as shown in Figure 8. The action priority is inferred using the overall QoS, importance of QoS and cost of service substitution. As an example marked in the figure: with a very low QoS, high importance of QoS and low cost of substitution, the adaptation strategy should be a service replacement. Another example, when the QoS is in a medium range, its importance is low, and the cost of substitution is high, then action priority is low. This means that an internal renegotiation action will be done between the involved parties. Although a deviation is done from the quality ranges in the contract, but the service substitution is not a wise decision due to the low importance of the QoS and high cost of substitution. Other similar decisions could be inferred by the same reasoning.

The plot of the output surface of the system shows the dependency of the output on the inputs. Figure 9 displays the dependency of overall QoS based on availability and response-time parameters. The figure perfectly shows that the QoS is very high when the availability is high and the response-time is low, and the QoS is very low when the response-time is high and availability is low. Figure 10 depicts the dependency of actionPriority based on the importance of QoS and cost of service substitution. The figure shows that when the QoS is low, the cost of substitution can determine the adaptation strategy. The high cost results

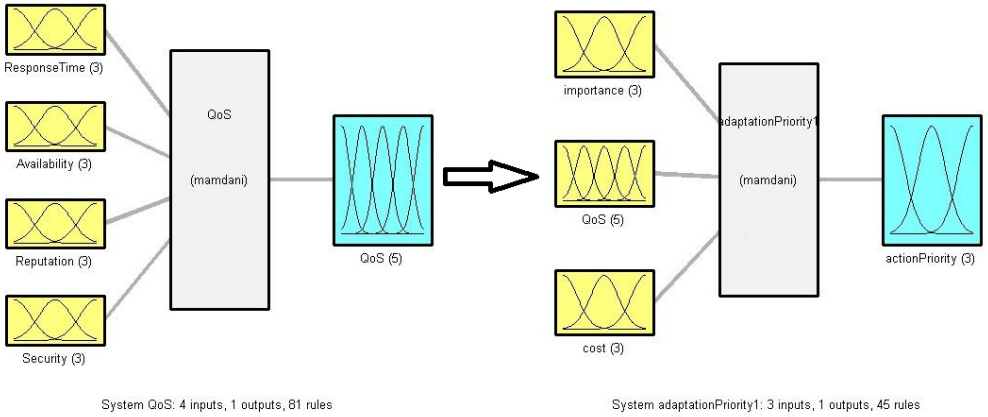


Figure 6. Approach design using fuzzy inference systems

14. If (ResponseTime is Good) and (Availability is Low) and (Reputation is Low) and (Security is High) then (QoS is Medium) (1)
15. If (ResponseTime is Good) and (Availability is Low) and (Reputation is Low) and (Security is Low) then (QoS is Medium) (1)
16. If (ResponseTime is Good) and (Availability is High) and (Reputation is Low) and (Security is Medium) then (QoS is High) (1)
17. If (ResponseTime is Good) and (Availability is High) and (Reputation is Medium) and (Security is Low) then (QoS is High) (1)
18. If (ResponseTime is Good) and (Availability is Medium) and (Reputation is Low) and (Security is Medium) then (QoS is Medium) (1)
19. If (ResponseTime is Good) and (Availability is Medium) and (Reputation is Medium) and (Security is Low) then (QoS is Medium) (1)
20. If (ResponseTime is Good) and (Availability is Low) and (Reputation is Low) and (Security is Medium) then (QoS is Medium) (1)
21. If (ResponseTime is Good) and (Availability is Low) and (Reputation is Medium) and (Security is Low) then (QoS is Medium) (1)
22. If (ResponseTime is Good) and (Availability is Low) and (Reputation is High) and (Security is Medium) then (QoS is Medium) (1)
23. If (ResponseTime is Good) and (Availability is Medium) and (Reputation is High) and (Security is Low) then (QoS is Medium) (1)
24. If (ResponseTime is Good) and (Availability is Medium) and (Reputation is Low) and (Security is High) then (QoS is Medium) (1)
25. If (ResponseTime is Good) and (Availability is Low) and (Reputation is Medium) and (Security is High) then (QoS is Medium) (1)
26. If (ResponseTime is Good) and (Availability is Medium) and (Reputation is Low) and (Security is Low) then (QoS is Medium) (1)
27. If (ResponseTime is Good) and (Availability is Low) and (Reputation is Medium) and (Security is Medium) then (QoS is Medium) (1)
28. If (ResponseTime is Medium) and (Availability is High) and (Reputation is High) and (Security is High) then (QoS is Very_High) (1)
29. If (ResponseTime is Medium) and (Availability is High) and (Reputation is High) and (Security is Medium) then (QoS is Very_High) (1)
30. If (ResponseTime is Medium) and (Availability is High) and (Reputation is Medium) and (Security is High) then (QoS is Very_High) (1)
31. If (ResponseTime is Medium) and (Availability is Medium) and (Reputation is High) and (Security is High) then (QoS is High) (1)
32. If (ResponseTime is Medium) and (Availability is High) and (Reputation is Medium) and (Security is Medium) then (QoS is High) (1)
33. If (ResponseTime is Medium) and (Availability is Medium) and (Reputation is Medium) and (Security is High) then (QoS is High) (1)

Figure 7. QoS-related Rules

in medium actionPriority which means renegotiation is a dominant strategy, while the decrease of the cost leads to higher action priority where substitution is preferred. It can be seen from the figure that actionPriority is very low when the QoS is high, regardless to the value of cost. Therefore, do-nothing strategy will be selected for minor deviations.

We evaluated the efficiency of our fuzzy adaptation approach in comparison with a naive Non-fuzzy approach. A non-fuzzy approach may perform a service substitution whenever a deviation occurs from the predefined ranges of parameters. We performed 8 experiments, with injecting a sample data per millisecond. A comparison of the approach with the Non-fuzzy one, with respect to number of service substitutions is illustrated in Figure 11. In general, the experiments show that our approach intelligently reduces the number of service substitutions in comparison to a naive adaptation method. This is done by a trade-off between overall degree of QoS, importance and cost of adaptation.

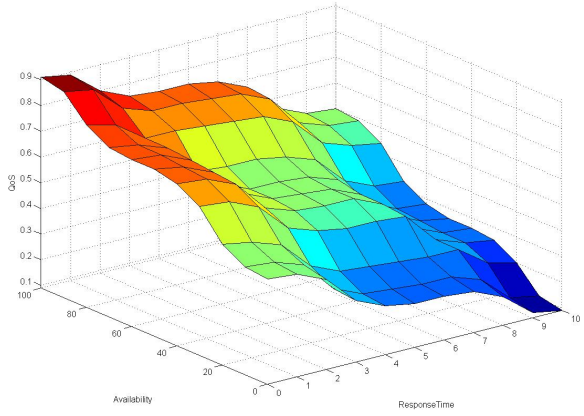


Figure 9. The plot surface of QoS

1. If (importance is Low) and (QoS is Very_Low) and (cost is Low) then (actionPriority is Substitution) (1)
2. If (importance is Low) and (QoS is Very_Low) and (cost is Medium) then (actionPriority is Renegotiation) (1)
3. If (importance is Low) and (QoS is Very_Low) and (cost is High) then (actionPriority is Renegotiation) (1)
4. If (importance is Medium) and (QoS is Very_Low) and (cost is Low) then (actionPriority is Substitution) (1)
5. If (importance is Medium) and (QoS is Very_Low) and (cost is Medium) then (actionPriority is Substitution) (1)
6. If (importance is Medium) and (QoS is Very_Low) and (cost is High) then (actionPriority is Renegotiation) (1)
7. If (importance is High) and (QoS is Very_Low) and (cost is Low) then (actionPriority is Substitution) (1)
8. If (importance is High) and (QoS is Very_Low) and (cost is Medium) then (actionPriority is Substitution) (1)
9. If (importance is High) and (QoS is Very_Low) and (cost is High) then (actionPriority is Substitution) (1)
10. If (importance is Low) and (QoS is Medium) and (cost is Low) then (actionPriority is Renegotiation) (1)
11. If (importance is Low) and (QoS is Medium) and (cost is Medium) then (actionPriority is Renegotiation) (1)
12. If (importance is Low) and (QoS is Medium) and (cost is High) then (actionPriority is Renegotiation) (1)
13. If (importance is Medium) and (QoS is Medium) and (cost is Low) then (actionPriority is Substitution) (1)
14. If (importance is Medium) and (QoS is Medium) and (cost is Medium) then (actionPriority is Renegotiation) (1)
15. If (importance is Medium) and (QoS is Medium) and (cost is High) then (actionPriority is Renegotiation) (1)
16. If (importance is High) and (QoS is Medium) and (cost is Low) then (actionPriority is Substitution) (1)
17. If (importance is High) and (QoS is Medium) and (cost is Medium) then (actionPriority is Renegotiation) (1)
18. If (importance is High) and (QoS is Medium) and (cost is High) then (actionPriority is Renegotiation) (1)
19. If (importance is Low) and (QoS is Very_High) and (cost is Low) then (actionPriority is Nothing) (1)
20. If (importance is Low) and (QoS is Very_High) and (cost is Medium) then (actionPriority is Nothing) (1)

Figure 8. Adaptation-related Rules

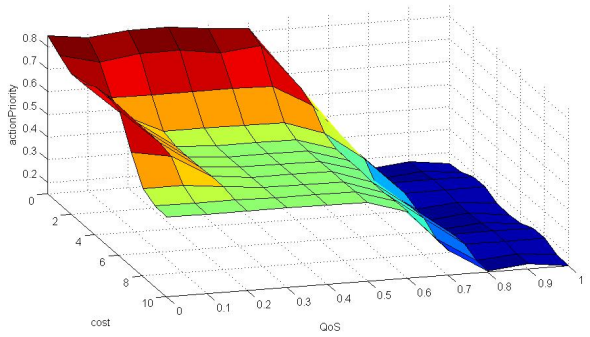


Figure 10. The plot surface of adaptation priority

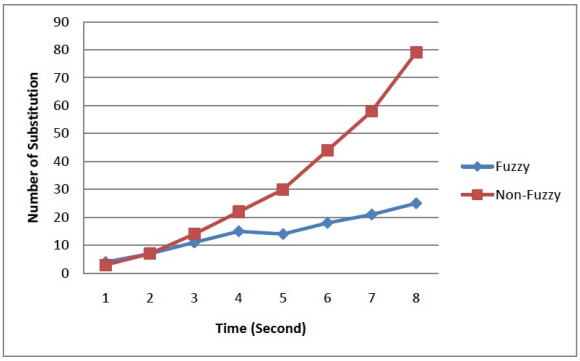


Figure 11. Comparing the efficiency of fuzzy with Non-fuzzy approach regarding the number of service substitutions

This way, minor deviations could be ignored due to high cost of service adaptation and partial satisfaction of QoS parameters are allowed. The figure shows that the fuzzy approach is very efficient in longer time and with higher sample data. The non-fuzzy approach exponentially increase the number of service substitution in this situation.

VI. Conclusions and Future Work

In this paper we have presented a novel approach for selecting adaptation strategies in SBAs based on fuzzy logic. We have shown how to model QoS parameters and associate them to adaptation strategies. We used hierarchical fuzzy inference systems for this purpose. Firstly, the degree of QoS can be inferred by changes of QoS parameters using a fuzzy inference engine. Second, a decision making for adaptation can be inferred by considering overall QoS, its importance and cost of substitution. Our approach allows partial satisfaction of QoS attributes by measuring their degree of membership using various membership functions provided for each parameter. Moreover, adaptation actions are taken by trading off between the degree of QoS, its importance and cost of service substitution. Therefore, the approach is able to intelligently choose between renegotiation or substitution of web services, which result in reducing the number of service substitutions. We particularly used hierarchy fuzzy systems for the problem of rule explosion and to increase the scalability and reusability of applied rules.

Our approach is flexible in that it is application independent and can support adaptation with respect to changes of parameters from various categories. However, only QoS parameters is investigated in this paper. As for the future work, we intend to consider other parameters (e.g context or CPU) and study their relation with the adaptation behaviour of the system. It is expected that increasing the number and category of parameters makes the fuzzy approach more applicable due to high complexity of finding a mathematical

relation between all the parameters.

In this work, we assumed that all parameters are able to be fuzzified. However, there may be parameters that are requested to be measured precise approaches. In this case, we would like to combine a fuzzy and non-fuzzy approach in order to efficiently consider all parameters in the decision making process. We also intend to consider the user preferences over each parameter in order to provide weight of parameters prioritize the importance of them. Furthermore, we would like to rank the rules accordingly. In this paper, we considered the same rank for all the rules and applied them in parallel.

Despite the efficiency of our approach in reducing the number of substitution, there is still lack of comprehensive comparison with a non-fuzzy approach in dealing with QoS management and adaptation decision. In general, a comparison between existing hard computing techniques and one of the soft computing techniques, say fuzzy logic, would be of great importance for the service research community.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

REFERENCES

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, pp. 38–45, 2007.

[2] A. Bucchiarone, C. Cappiello, E. D. Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore, "Design for adaptation of service-based applications: main issues and requirements," in *Proc. ICSOC/ServiceWave*, 2009, pp. 467–476.

[3] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

[4] ——, "The concept of a linguistic variable and its application to approximate reasoning," *Inf. Sci.*, vol. 8, no. 3, pp. 199–249, 1975.

[5] J. Harney and P. Doshi, "Adaptive web processes using value of changed information," in *Proc. ICSOC'06*, 2006, pp. 179–190.

[6] G. Chafle, P. Doshi, J. Harney, S. Mittal, and B. Srivastava, "Improved adaptation of web service compositions using value of changed information," in *Proc. ICWS'07*, 2007, pp. 784–791.

[7] R. Kazhamiakin, M. Pistore, and A. Zengin, "Cross-layer adaptation and monitoring of service-based applications," in *Proc. ICSOC/ServiceWave Workshops*, 2009, pp. 325–334.

[8] M. Comuzzi and B. Pernici, "A framework for QoS-based web service contracting," *ACM Transactions on the Web*, vol. 3, no. 3, June 2009.

[9] V. Andrikopoulos, M. Fugini, M. P. Papazoglou, M. Parkin, B. Pernici, and S. H. Siadat, "QoS contract formation and evolution," in *Proc. EC-Web'10*, 2010, pp. 119–130.

[10] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "QoS-aware replanning of composite web services," in *Proc. ICWS'05*, 2005, pp. 121–129.

[11] R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann, "Adaptation of service-based applications based on process quality factor analysis," in *Proc. ICSOC/ServiceWave*, 2009, pp. 395–404.

[12] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, 2004.

[13] P. Li, M. Comerio, A. Maurino, and F. D. Paoli, "An approach to Non-functional property evaluation of web services," in *Proc. ICWS'09*, 2009, pp. 1004–1005.

[14] F. Islam, M. N. Lucky, and B. Pernici, "Business analysis of web service repairability," in *Proc. RCIS'10*, 2010, pp. 463–472.

[15] B. Qiu, "The application of fuzzy prediction for the improvement of QoS performance," in *Proc. IEEE International Conference on Communications*, vol. 3, Jun. 1998, pp. 1769 –1773.

[16] C. Koliver, K. Nahrstedt, J.-M. Farines, J. D. S. Fraga, and S. A. Sandri, "Specification, mapping and control for QoS adaptation," *Real-Time Syst.*, vol. 23, pp. 143–174, July 2002.

[17] B. Li and K. Nahrstedt, "A control-based middleware framework for quality of service adaptations," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1632–1650, 1999.

[18] C. Koliver, J.-M. Farines, and K. Nahrstedt, "QoS adaptation based on fuzzy theory," in *Soft Computing in Communications*, L. Wang, Ed., 2004, pp. 245–265.

[19] R. Cheung, J. Cao, G. Yao, and A. T. S. Chan, "A fuzzy-based service adaptation middleware for context-aware computing," in *Proc. EUC'06*, 2006, pp. 580–590.

[20] S.-N. Chuang and A. T. S. Chan, "Dynamic QoS adaptation for mobile middleware," *IEEE Trans. Softw. Eng.*, vol. 34, pp. 738–752, November 2008.

[21] J. O'Sullivan, D. Edmond, and A. Ter Hofstede, "What's in a service?" *Distrib. Parallel Databases*, vol. 12, pp. 117–133, September 2002.

[22] A. Paschke and E. Schnappinger-Gerull, "A categorization scheme for SLA metrics," in *Proc. Service Oriented Electronic Commerce*, 2006, pp. 25–40.

[23] R. R. Yager, "On the construction of hierarchical fuzzy systems models," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 28, no. 1, pp. 55 –66, Feb. 1998.

[24] M.-L. Lee, H.-Y. Chung, and F.-M. Yu, "Modeling of hierarchical fuzzy systems," *Fuzzy Sets Syst.*, vol. 138, pp. 343–361, September 2003.

Preventing KPI Violations in Business Processes based on Decision Tree Learning and Proactive Service Substitution

Branimir Wetzstein* Asli Zengin† Raman Kazhamiakin† Marco Pistore† Dimka Karastoyanova*

*Institute of Architecture of Application Systems
University of Stuttgart, Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

†Fondazione Bruno Kessler IRST
Trento, Italy
{zengin, raman, pistore}@fbk.eu

***Abstract*—The performance of business processes implemented as service-based applications is measured and monitored in terms of Key Performance Indicators (KPIs). If monitoring results show that the KPI targets are violated, the underlying reasons have to be identified and the process should be adapted accordingly to remove the violations. In this paper we propose an integrated monitoring, prediction and adaptation approach for preventing KPI violations of business process instances. KPIs are monitored continuously while the process is executed. Based on KPI measurements of historical process instances we use decision tree learning to construct classification models which are then used to predict the KPI value of an instance while it is still running. If a KPI violation is predicted, we identify adaptation requirements and adaptation strategies consisting of service substitutions in order to prevent the violation. The approach has been implemented and experimentally evaluated.**

***Keywords*-Business Activity Monitoring, Business Process Intelligence, WS-BPEL, Decision Tree, Process Adaptation**

I. Introduction

In recent years, the industry experienced a wide adoption of the service-oriented architecture for the implementation of business processes. Service-based applications realize such processes by modeling and deploying a complex, distributed and layered system, where the business model of an application is implemented through a service composition which orchestrates services running on top of service infrastructures. To be effective, such applications should meet certain business goals, traditionally expressed as Key Performance Indicators (KPIs) of the business processes. These KPIs are continuously monitored at run-time using business activity monitoring techniques.

If monitoring shows that KPI targets are not reached, then firstly it is necessary to identify the factors which strongly influence the KPI and often cause KPI target violations. In complex business processes, the relations between the overall business process performance and lower-level influential factors and their combination are neither explicit nor easy to reveal. Secondly, in addition to identifying the influential factors based on historical process executions, it is desirable to be able to predict for a running process instance whether it will reach the KPI target. Finally, if a KPI target violation is predicted, one has to identify adaptation strategies which can potentially improve the performance of the process instance, thus preventing the violation.

In this paper we present an integrated monitoring, analysis, prediction and adaptation approach that aims to address the above problems. The execution of the business process is continuously monitored based on runtime events published by the process execution middleware. Based on monitoring data of historical process instances, we use decision tree algorithms in order to learn the dependencies between the KPI and the influencing lower-level metrics. The resulting decision tree is used for KPI prediction for future process instances. If for a running process instance, a KPI target violation is predicted, adaptation requirements are extracted from the decision tree which specify conditions on metric values which should be improved. In the next step, we identify adaptation strategies consisting of adaptation actions which should be performed in order to satisfy the adaptation requirement conditions. In this paper we thereby focus on service substitution as the only adaptation action type. In a subsequent step we then filter and rank adaptation strategies based on constraints and preferences considering the QoS characteristics of the services to be substituted. Finally, the process instance is pro-actively adapted in order to prevent the KPI violation. The presented work builds on the work presented in [1] which focused on monitoring and KPI dependency analysis, and extends, refines and evaluates our preliminary ideas presented in [2], where the overall monitoring, analysis and adaptation framework has been described at a higher level.

The paper is organized as follows. We begin with a motivating scenario that describes the problem and which we use in the rest of the paper to present our solution. Section III gives an overview of the approach. Sections IV and V describe the prediction and adaptation aspects in detail. Section VI describes the implementation of the approach and presents results of an experimental evaluation. Finally, we give a summary of related work and conclude the paper together with the directions for future work.

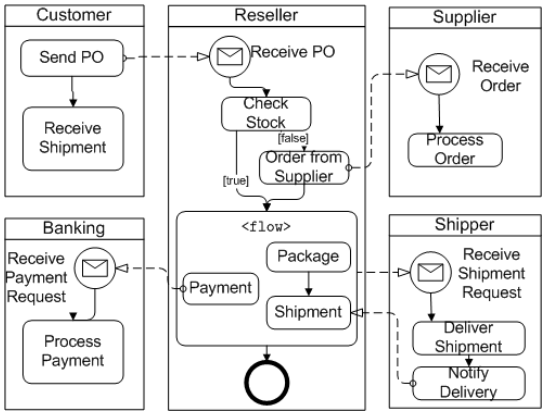


Figure 1. Purchase Order Process

II. Scenario and Motivation

In this section we introduce a scenario that we use in the following sections for explaining our approach. As shown in Figure 1, the scenario consists of a purchase order process implemented by a reseller who offers products to its customers and interacts with external suppliers, banking, and shipment services for processing the order. Furthermore, the reseller uses warehouse and customer services, which are internal to the organization.

For measuring the performance of its business process, the reseller defines a set of Key Performance Indicators (KPIs). For instance in this scenario a typical KPI is "order fulfillment lead time" (process duration from order receipt until shipment arrives at the customer) or "order delivered in full and in time" (see Supply Chain Operations Reference Model[1]). In addition to the KPI metric measurement specification, a KPI definition includes also a target value function which specifies KPI classes based on a business goal, e.g., order fulfillment lead time < 3 days is "good", < 5 days is "medium", and otherwise "bad".

After defining the KPIs, they have to be measured based on executed process instances. If after a while the monitoring shows an unsatisfactory result, i.e., the KPI targets are violated for many instances (i.e. purchase orders are late, in our case), the reseller wants to find out the influential factors which lead to good or bad KPI values. Understanding the reasons why certain orders are delivered on time and others are not, is often not trivial, as the KPI depends on the combination of several factors such as ordered product types and amounts (input data of the process), duration and availability of the internal services, duration, reliability and SLA conformance of external services etc. For example, standard shipment duration could take from one to five days or in exceptional cases even longer, supplier delivery time might depend on certain product types and amounts. These deviations in service behavior lead to the different outcomes of process instances considering KPI targets.

After the influential factors for KPI violations based on a set of a past instances have been understood, the next logical step is to try to prevent the violations in future. This can be done using runtime process instance adaptation. Thereby, in this paper we focus on service substitution as adaptation mechanism. Therefore, we assume that there is a set of candidate services for a set of service types used in the process. E.g, there might be several alternative shippers which offer different service levels via shipment options (e.g., standard, premium, overnight express); each of those options can be modeled as a candidate service with different quality of service (QoS) characteristics (such as shipment delivery time, shipment cost, reputation, etc.)

III. Overview of the Approach

In this section we give an overview of our approach by describing its lifecycle as shown in Figure 2 . The supporting architecture and implementation will be described in Section VI. The lifecycle consists of the following phases:

1. Monitoring: In the monitoring phase, all metrics specified in the *metrics model* are monitored. That includes the KPI metrics but also lower-level metrics of the potential influential factors. As a result, *metric values* for a set of executed process instances are stored and provided to the next phase.

2. KPI Dependency Analysis: After a certain number of executed process instances, based on a *KPI analysis model* a decision tree is trained which helps to understand when a KPI is violated. The KPI analysis model contains the definition of the KPI and specifies the subset of the metrics from the metrics model which should be used for the analysis. The resulting decision tree serves from now on as a *classification model* for future process instances and is used for prediction (Section IV-A).

3. KPI Prediction: A *checkpoint model* specifies at which points in the process the KPI prediction should take place. When a running process instance reaches a checkpoint, it halts its execution. The metric values which have been measured until the checkpoint for that instance are gathered and used as input to the classification model learned in phase 2 (Section IV-C). The prediction result is either a predicted KPI class (e.g., "green", "yellow" or "red") or a reduced tree, which shows which metrics should be improved to reach a specific KPI class and serves thus as basis for adaptation.

4. Identification of Adaptation Requirements and Adaptation Strategies: Adaptation requirements are identified by extracting metrics which should be improved from the reduced tree (Section V-A). Based on the adaptation requirements, a set of alternative adaptation strategies is identified by taking into account available adaptation actions, i.e. in our case, service substitutions. An adaptation strategy thus consists of a set of service candidates which should be

[1] http://www.supply-chain.org/

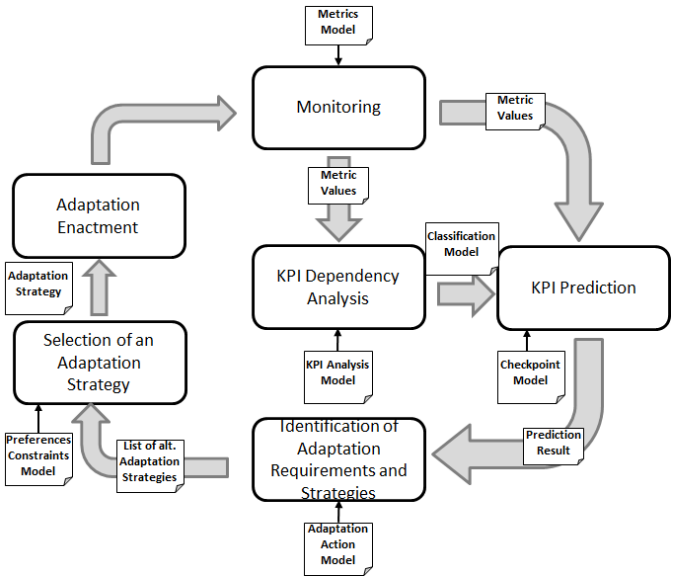


Figure 2. Lifecycle of the Approach

used in the process instance in order to reach a certain KPI class.

5. Selection of an Adaptation Strategy: The list of alternative adaptation strategies is filtered and ranked based on a *constraints and preferences model*. In this paper, we use maximal cost of the substituted services which an adaptation strategy has to satisfy as a constraint. Preferences are specified as weights on the QoS characteristics cost, duration, and reputation which enables ranking of strategies according to scores.

6. Adaptation Enactment: The first ranked adaptation strategy is enacted. This is done by binding selected services to the process instance. The process instance is unblocked and continues its execution.

While steps 3-6 are being performed for a certain number of instances, the monitoring (step 1) is continued. After a certain number of instances, the effectiveness of the adaptations is evaluated by checking how many KPI violations have been prevented and how many instances still violate their KPIs. This might lead to adjustment of the models, e.g., adjustment of KPI targets, (re)moving of checkpoints, adjustment of the constraints and preferences model.

IV. Monitoring and Prediction of KPIs

In the following we first give some background information on KPI dependency analysis based on decision trees, and then explain how those trees are used for KPI prediction.

A. Background on KPI Dependency Analysis

A KPI is defined as a tuple consisting of a metric definition and a target value function which maps value ranges of the metric to categorical values (KPI classes). For example, if we choose "order fulfillment lead time" as the KPI metric, then we could specify the function: m $<$ 4 days is "good", 4 days $<$ m $<$ 7 days is "medium", otherwise "bad". If the KPI is used for defining a service level objective then typically a binary function is used (with the classes "violated" and "fulfilled").

The KPI class is evaluated per process instance. It depends typically on the combination of a set of influential factors, e.g., input data to the process (ordered product types and amounts), service outputs and durations (e.g., shipment delivery time). In order to find out the dependencies of the KPI class and these factors, we use classification learning known from machine learning and data mining [3]. In a classification problem, a dataset is given consisting of a set of examples (process instances) described in terms of a set of explanatory attributes (influential factor metrics) and a categorical target attribute (KPI class). The explanatory attributes, also called predictive variables, may be partly categorical and partly numerical. By using a learning algorithm, based on the example set (training set) a classification model is created, whose purpose is to identify recurring relationships among the explanatory variables which describe the examples belonging to the same class. The goal is to *explain* past classifications and also to *predict* the class of examples for which only the values of the explanatory attributes are known.

It has already been shown in [1] how decision trees can be used for explanation purposes, i.e. to explain how KPI classes depend on a set of influential factor metrics (a.k.a. KPI dependency trees); in this paper we utilize those trees for prediction. There are different types of algorithms for classification model learning and prediction, e.g., artificial neural networks, classification rules, and support vector machines [3]. We have decided to use decision trees in our context because of their following advantages. They constitute a white box model as they show explicitly the relationships between explanatory attribute value ranges and KPI classes. Thus they are easy to understand and interpret for people and enable human support in the learning and adaptation phases. In particular they support extraction of adaptation requirements from the tree paths (Section V-A). Furthermore, decision trees support both numeric (typically, time based metrics) and categorical explanatory attributes (typically process data based metrics).

B. Modeling for Prediction

Prediction for a running process instance is performed at a checkpoint [4]. One can define several checkpoints per process model. We define a checkpoint as consisting of (i) a trigger which is a process runtime event (typically signaling start or completion of an activity; the event is typically configured to be blocking, i.e. to stop process instance execution until prediction and potential adaptation are performed), (ii) a KPI analysis model (KPI definition and a set of influential factor metrics (representing the features in classification learning) from the metrics model), (iii) a

reference to an adaptation action model and (iv) a reference to a constraints and preferences model. The influential factor metric set consists of the *known metrics* until the checkpoint and a set of *unknown metrics* at the checkpoint which however are "adaptable" by the available adaptation actions. At the warehouse "order in stock" check at the beginning of the process, known metrics are for example the ordered product types and amounts, the customer, and the process duration until that activity. Unknown but adaptable metrics are for example supplier delivery time and shipment delivery time. The goal of including unknown metrics into the classification learning is to enable understanding which adaptations are needed for preventing a violation (Section V-A).

C. Runtime Prediction based on Decision Trees

At process runtime, after a sufficiently large set of instances has been executed and monitored, based on the checkpoint definition, for each checkpoint a decision tree is learned (see [1] for more information on monitoring and KPI dependency analysis for explanation purposes). It explains how the KPI classes of those history instances depend on influential factor metrics. An example is shown in Figure 3. The tree has been generated using the J48 algorithm (see Section VI and [3]) for a checkpoint defined right after the warehouse "orderInStock?" check at the beginning of the process. The tree contains known metrics (orderInStock, itemQuantity) and unknown metrics at the checpoint (shipment delivery time, supplier delivery time). It shows, for example, that for "order in stock=true", and "item quantity $\leq$ 20" 30 process instances have reached the KPI class "green". Quality aspects of the generated classification model are discussed in the evaluation section VI.

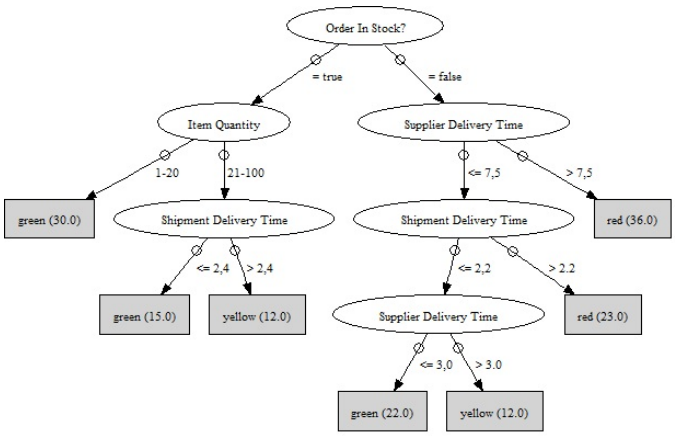


Figure 3. KPI Dependency Tree for Order Fulfillment Lead Time after In Stock Check

In the next step the decision tree can be used for prediction. When the process instance execution reaches a checkpoint which is signaled by the specified event, the known metrics for that instance until the checkpoint are gathered and "inserted" into the decision tree for that checkpoint. Therefore we traverse the tree breadth-first; if the current node corresponds to a known metric, we follow the outgoing branch whose condition is satisfied by the measured metric value and replace the current node with the target node of that branch; otherwise, if the metric is unknown we leave the node in the tree (and continue with its children until a leaf node is reached). As the result we get a subtree of the original one (in the following denoted as predicted tree) consisting either of (1) just one leaf representing the prediction of the corresponding KPI class; (2) a tree containing one or more nodes which correspond to unknown metrics which are adaptable by the available adaptation actions. Figure 4 shows an example instantiation of the tree from Figure 3 assuming that for the process instance we have measured "orderInStock=false".

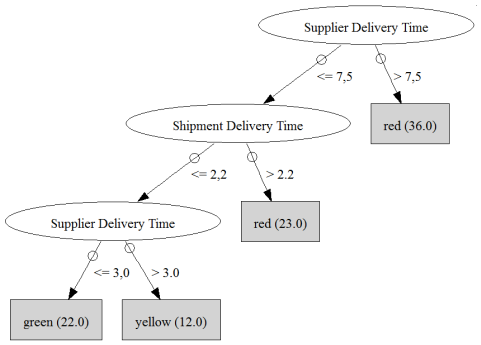


Figure 4. Instantiated Tree for Order In Stock = False

V. Adaptation based on Prediction

In the following we describe the identification of adaptation requirements based on prediction results, the identification of adaptation strategies, and selection and enactment of one of those strategies.

A. Identification of Adaptation Requirements

After the predicted tree is generated, we have to decide whether adaptation is needed, and if yes, which influential metrics should be improved and how. The predicted tree shows how the KPI class of the running instance depends on the unknown adaptable metrics.

If the predicted tree contains only one leaf denoting the KPI class, then the predicted KPI class is independent of the adaptable metrics, and thus independent of which services are invoked. In case of a good KPI class ("green") simply the default services can be invoked (i.e., services which are optimal according to the specified preferences and constraints model). In case of a medium or bad KPI class, an adaptation which leads to a better KPI class is not possible, at least according to history data. In that case, one could send notifications to affected users or customers, or one could still

Adaptation Requirements		**Adaptation Strategies**				
Conditions	KPI Class	Strategy	QoS score	Constraints	Score	Rank
Shipment Delivery Time $<$ 2,2 Supplier Delivery Time $<$ 3,0	green (0.65)	Sh1 Premium + Su1 Premium	0.25	ok	0.16	1
		Sh1 Premium + Su2 Premium	0.19	ok	0.12	3
		Sh2 Premium + Su1 Premium	0.15	nok	0.10	-
		...	...	...	...	
Shipment Delivery Time $<$ 2,2 Supplier Delivery Time $<$ 7,5	yellow (0.35)	Sh1 Premium + Su1 Standard	0.38	ok	0.13	2
		Sh1 Premium + Su2 Standard	0.28	ok	0.10	4
		Sh1 Premium + Su1 Premium	0.25	ok	0.09	5
		...	...	...	...	

Table I
RANKING OF ADAPTATION STRATEGIES

try to select better services to improve "as much as possible", however, this is out of scope of our current approach.

If the predicted tree contains more than just one leaf (as the one in Figure 4), then the non-leaf nodes correspond to influential factor metrics which are adaptable by the predefined adaptation actions. It means that in history instances those adaptable metrics values have varied in such a way that different KPI classes have been produced. For example, the example tree shows that each time a supplier delivery time was above 7,5 days, the resulting KPI class was "red". That happened for 36 instances. If on the other hand we could ensure a supplier delivery time below 3 days and a shipment delivery time below 2,2 days we would very likely (assuming that the classification model has a high accuracy, see Section VI) reach a "green" KPI class. The idea towards adaptation is thus (1) to extract those paths and the corresponding metric conditions, which lead to acceptable KPI classes, and then (2) select adaptation actions, which will lead to satisfaction of those metrics conditions.

We call the paths of the predicted tree which lead to acceptable KPI classes *safe paths*. The user can configure which KPI classes are acceptable for an instance (e.g., "green", and "yellow" could be safe, while "red" is to be avoided) and assign a weight to each KPI class which is needed later for the calculation of a score of an adaptation strategy.

If we ensure one of the safe paths, then we avoid all of the bad paths. Thus, eventually each safe path is an alternative *adaptation requirement*. An adaptation requirement (AR) is defined as a tuple consisting of the target KPI class and a set of metric conditions which should be achieved in order to reach the KPI class. An adaptation requirement is extracted from a safe path as follows: from each branch on the path we extract the metric value condition and add it to the adaptation requirement. The conditions are combined by using logical conjunction, i.e., all conditions have to be true in order to satisfy the requirement. If on the path there are metrics of the same type, then we combine their value ranges. Finally, in the last step if there are conditions which are satisfied with semantically worse metric values (e.g., if a condition is "supplier delivery time $>$ 2 days"), then it can be ignored and removed from the requirement because the value does not have to be improved.

As a result we get a set of alternative adaptation requirements each consisting of a conjunction of conditions over adaptable metrics which have to be achieved. In our case, adaptable metrics are QoS characteristics of candidate services. For the example tree (Figure 4), we can extract two adaptation requirements as shown in Table I (first two columns), one for the KPI class "green" (with weight 0.65) and one for the KPI class "yellow" (weight 0.35).

B. Identification of Adaptation Strategies

After the requirements have been identified, the next step is to identify adaptation strategies which can be used to satisfy the adaptation requirements. An *adaptation strategy (AS)* consists of a set of *adaptation actions* which satisfy the conditions of an adaptation requirement. In our case, adaptation actions are service substitutions which bind new candidate services to the process instance which according to their QoS characteristics should satisfy the adaptation requirement conditions. For example, if an AR condition specifies that supplier delivery time should be below 3 days, we want to select a supplier service that can reach that target according to its QoS characteristics. For new services the QoS characteristics could be derived from SLAs (where some max or avg values are guaranteed); for services which already have been used in many instances, the QoS characteristics can be derived from measurements.

In the first step, for each adaptation requirement a set of alternative strategies is identified by simply enumerating and combining all possible service candidates which satisfy the conditions in the adaptation requirement. The result is a set of alternative adaptation strategies which all according to history measurements and the QoS characteristics of service candidates would lead to an acceptable KPI class.

C. Ranking of Adaptation Strategies

In the second step, that set is further filtered and ranked according to the constraints and preferences model. We define constraints as predicates over properties of selected services and/or metrics measured so far for the running instance (e.g., maximal cost of supplier and shipper service $<$ x). If a constraint evaluation evaluates to false for a strategy, then that strategy is removed from the set.

For ranking of strategies, we first calculate the QoS scores of the services used in the strategy. For each selected candidate service in an AS its QoS score is calculated by aggregating the QoS properties using the Simple Additive Weighting - Multiple Criteria Decision Making Approach for QoS aggregation [5]. Each of the three QoS characteristics we use (duration, cost, reputation) is assigned a weight (sum of all weights must be equal to 1). The result of this procedure, is a score by which the overall QoS of a service candidate can be compared with the overall QoS of another service candidate. If there is more than one service substitution per strategy (e.g., if we need to replace both the supplier and shipment), we simply sum up scores of all services, for the time being ignoring process structure for simplicity.

In addition to the QoS score, we have to take into account that adaptation requirements could have resulted from paths with different KPI classes. Thus, the overall score of a strategy is determined by multiplying the KPI class weight and the QoS score as shown in Table I. Finally, the best strategy is selected and enacted. In case of a service substitution, enactment involves binding a new service to the process instance replacing an existing, default service binding.

VI. Implementation and Experimental Evaluation

We have implemented the approach as shown in Figure 5. Our prototype uses Apache ODE[2] as the business process execution engine. The engine has been extended to support blocking events which stop process instance execution needed for prediction and instance adaptation[6]. The monitoring is performed based on the ESPER complex event processing (CEP) framework[3] which calculates metrics based on events which are published by the process engine and a QoS monitor as already described in [1]. The classification model learner is based on the WEKA suite[4] which provides decision tree algorithm implementations. The rest is implemented in Java. Service substitution has been implemented transparent to the BPEL process using an AOP based approach [6]. All phases are supported by a GUI (dashboards) which allow the user to change settings and model parameters and see the results.

A. Experimental Evaluation

We have implemented the scenario from Section 2 as a BPEL process interacting with six Web services. The Web

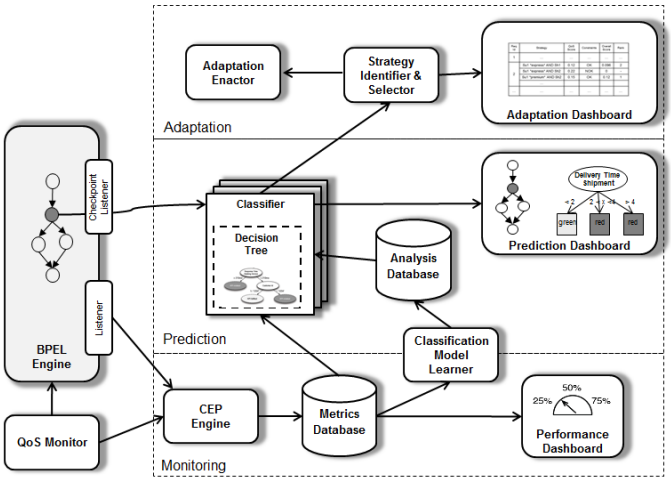


Figure 5. Architecture of the Prototype

services have been implemented in Java and for experimental purposes simulate certain influential factors (e.g., duration and output are made dependent on factors such as product types and amounts, and random behavior). For experimentation, we have deployed all these components on a single desktop PC.

We define Order Fulfillment Lead Time as the KPI to be analyzed and define two checkpoints in the process (after "Warehouse" check (i.e., both supplier and shipper can still be selected), and before "Shipment" thus allowing only the shipper to be selected). We create a set of overall 30 service candidates with different QoS characteristics (specified as mean values) and create a configuration which simulates the behavior of those services according to their QoS characteristics, but with deviations .

Learning Phase: We trigger the execution of 500 process instances using a test client. During process instance execution, the previously specified metrics are measured and saved in the metrics database. Then, for each checkpoint a decision tree is learned. The performance of the learning of the trees is about 15 seconds for 500 instances (on a standard laptop computer) and increases linearly with the number of instances. As learning can be done in the background it does not affect the instance execution. For tree generation we have used the J48 algorithm. The quality of the trained tree as a classification model can be assessed in terms of its accuracy, i.e., the percentage of correctly classified instances (from a test set) by the model. This metric is provided by the decision tree algorithm after validation of the model (cross-validation). The accuracy depends among other factors on the number of instances available for learning, and the selection of explanatory attributes and how good they describe the KPI class. In general, the later the checkpoint is defined in the process the better the tree quality will be, because there are more known metrics that can be used for training the

[2]http://ode.apache.org/
[3]http://esper.codehaus.org/
[4]http://www.cs.waikato.ac.nz/ml/weka/

tree (see column tree accuracy in Table II). In our approach, we assume that a classification model with a reasonably high accuracy has been created and do not take the accuracy into account in the following prediction and adaptation phases.

Prediction and Adaptation Phase: For the prediction and adaptation, we use two different constraints and preference models, one preferring lower cost, the other lower duration. For each model, we perform three experimental runs with 200 instances per run. The first run is performed using the default services (optimal according to the preferences model) without using the prediction and adaptation framework. In the other two runs, the prediction and (potential) prevention is performed at two different checkpoints. We evaluate for each instance what is predicted and whether the prediction has been correct ("measured"); this is done for the prediction types "No (Adaptation) Need" (predicted KPI class is "green" or "yellow"), "Too Late" (predicted KPI class is "red"), and "Adaptation Need". The results are shown in Table II.

The prediction and adaptation time together are below a second, thus making it only a factor for very short running processes. Firstly, the results show that the KPI performance (column "KPI Evaluation") has been considerably improved by using our framework (run 2 and 3 outperform run 1). For example, for the first preference model the number of violations (KPI class = "red") has been reduced from 64 to 49 and 23, respectively. Secondly, the prevention effectiveness depends on settings in the preferences model and is in our case obviously much better when the preference is set on duration rather than cost. This is because substituted services are not always behaving as expected from their specified QoS characteristics (i.e., not satisfying the corresponding AR condition). Thus, when choosing services which just so satisfy the AR condition, the risk of a violation of that AR condition is higher. Thirdly, the later the checkpoint is chosen, the higher the prevention effectiveness as the prediction accuracy is higher. On the other hand, there is an increasing risk that it is too late to adapt ("Too Late" column). Of course, for even better performance, we could predict and adapt at both checkpoints for each process instance.

VII. Related Work

Our approach is related to approaches from the area of business process intelligence which combine service composition monitoring and performance analysis techniques, and to approaches from the area of QoS-aware service composition and adaptation.

In the area of process performance monitoring and analysis, most closely related to our approach is iBOM [7] which is a platform for monitoring and analysis of business processes based on machine learning techniques. It supports similar analysis mechanisms as in our approach such as decision trees, but does not deal with adaptation, i.e., extraction of adaptation requirements from the decision trees and derivation of adaptation strategies as in our approach. [8] presents an integrated KPI monitoring and prediction approach which uses machine learning techniques for prediction. It supports not only instance level KPI prediction as in our approach but also time series based prediction across process instances. It however does not deal with adaptation. We do not exploit information on process structure during dependency analysis, as the approach described in [9], but rely on machine learning algorithms to find those dependencies supporting not only numerical but also data-based metrics.[4] deals with prediction of numerical metric values based on artificial neural networks and introduces the concept of a checkpoint used for prediction which we have reused in our approach. [10], [11] also cover the phases monitoring, prediction and adaptation as in our approach focusing on prevention of SLO violations by adapting the process via service substitution and fragment substitution, respectively. The best adaptation strategy is selected by performing a numerical KPI prediction for each adaptation strategy alternative separately and then selecting the best result. Our (analysis and) adaptation approach is different, as we use decision trees which as a white box classification model enable explicit extraction of adaptation requirements and strategies from the classification model. We also deal with ranking of strategies based on constraints and preferences.

There are several existing works in the context of QoS-aware service composition [5], [12] which describe how to create service compositions which conform to global and local QoS constraints taking into account process structure when aggregating QoS values of atomic services. We have reused concepts from those works when it comes to the definition of the constraints and preferences model and calculation of QoS scores [5]. Currently, we are simply enumerating all combinations of services when identifying adaptation strategies; that is only feasible for a small number of service types and could be optimized as described, for example, in [5]. Furthermore, these approaches can be used for QoS-based adaptation by replanning the service composition during monitoring [13]. In [14] the PAWS (Processes and Adaptive Web Services) framework is presented which takes into account local and global QoS constraints for selection of Web services at composition runtime. After designers have defined global and local QoS constraints, if at runtime a QoS requirement cannot be met, the framework chooses among a set of recovery actions such as retry, substitute, and compensate. Our approach is different in that we do not look at the process structure for finding the dependencies, but use machine learning techniques and also support process data-based metrics during analysis in addition to numerical metrics (such as duration and cost).

VIII. Conclusions and Future Work

In this paper we have presented an integrated monitoring, prediction and adaptation approach for preventing KPI violations in service compositions. We use decision trees

Learning		Prediction and Prevention (200 instances per run)				
Prediction Point	Decision Tree Accuracy/Size	QoS Weights time/cost/reput.	No Need (predicted/measured)	Too late (predicted/measured)	Adaptation Need (predicted/successful)	KPI Evaluation green/yellow/red
None	N/A	0.15/0.7/0.15	N/A	N/A	N/A	110/26/64
		0.5/0.3/0.2	N/A	N/A	N/A	148/31/21
Warehouse	88,2%/83	0.15/0.7/0.15	102/63	0	98/88	119/32/49
		0.5/0.3/0.2	108/105	0	92/90	183/12/5
Shipment	94,7%/77	0.15/0.7/0.15	85/85	6/6	109/92	157/20/23
		0.5/0.3/0.2	105/103	5/5	90/88	180/11/9

Table II
EXPERIMENTAL RESULTS

in order to learn how the KPI class depends on a set of influential factors. At checkpoints, the KPI class of the running process instance is predicted based on the learned decision tree and metric data gathered for that instance until the checkpoint. In order to prevent KPI violations, adaptation requirements are extracted from the tree and then a set of alternative adaptation strategies is identified which can satisfy those requirements. The identified adaptation strategies are filtered and ranked according to QoS constraints and preferences. We have implemented the approach and evaluated it based on a purchase order processing scenario.

In our future work we will extend the approach in several directions. Firstly, we will implement different types of adaptation actions (which can be used in adaptation strategies) beyond service substitution on different applications layers of a service-based application. In that context, one could think of infrastructural reconfigurations on service layer, changes in the control flow of service compositions at the service composition layer, and others. Secondly, we will in particular address the cross-layer aspect by looking at how adaptation actions on different layers influence each other, e.g., a reconfiguration of the infrastructure has an impact on all services and process instances running on that infrastructure. That has to be taken into account during identification and selection of adaptation strategies.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Communitys Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

REFERENCES

[1] B. Wetzstein, P. Leitner, F. Rosenberg, I. Brandic, S. Dustdar, and F. Leymann, "Monitoring and Analyzing Influential Factors of Business Process Performance," in *Proceedings of the IEEE International Enterprise Distributed Object Computing Conference (EDOC09)*, 2009, pp. 141–150.

[2] R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann, "Adaptation of Service-Based Applications Based on Process Quality Factor Analysis," in *Proceedings of the 2nd Workshop on Monitoring, Adaptation and Beyond (MONA+)*, 2009.

[3] I. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann Pub, 2005.

[4] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime Prediction of Service Level Agreement Violations for Composite Services," in *Proceedings of the 3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing (NFPSLAM-SOC'09)*, 2009.

[5] M. C. Jaeger, G. Mühl, and S. Golze, "QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms," in *OTM Conferences (1)*, 2005, pp. 646–661.

[6] D. Karastoyanova and F. Leymann, "BPEL'n'Aspects: Adapting Service Orchestration Logic," in *ICWS '09: Proceedings of the 2009 IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 222–229.

[7] M. Castellanos, F. Casati, U. Dayal, and M.-C. Shan, "A Comprehensive and Automated Approach to Intelligent Business Processes Execution Analysis," *Distributed and Parallel Databases*, vol. 16, no. 3, pp. 239–273, 2004.

[8] L. Zeng, C. Lingenfelder, H. Lei, and H. Chang, "Event-Driven Quality of Service Prediction," in *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC'08)*, 2008.

[9] L. Bodenstaff, A. Wombacher, M. Reichert, and M. Jaeger, "Monitoring Dependencies for SLAs: The MoDe4SLA Approach," in *Proceedings of the 2008 IEEE International Conference on Services Computing (SCC'08)*, 2008.

[10] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, "Monitoring, Prediction and Prevention of SLA Violations in Composite Services," in *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS'10)*, 2010.

[11] P. Leitner, B. Wetzstein, D. Karastoyanova, W. Hummer, S. Dustdar, and F. Leymann, "Preventing SLA Violations in Service Compositions Using Aspect-Based Fragment Substitution," in *Proceedings of the 8th International Conference on Service Oriented Computing (ICSOC 2010)*. Springer Berlin Heidelberg, December 2010, Conference Paper.

[12] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[13] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "QoS-Aware Replanning of Composite Web Services," in *ICWS*, 2005, pp. 121–129.

[14] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "PAWS: A Framework for Executing Adaptive Web-Service Processes," *IEEE Software*, vol. 24, no. 6, pp. 39–46, 2007.

LAYSI: A Layered Approach for SLA-Violation Propagation in Self-manageable Cloud Infrastructures

Ivona Brandic, Vincent C. Emeakaroha,
Michael Maurer, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Vienna, Austria
{*ivona,vincent,maurer,dustdar*}*@infosys.tuwien.ac.at*

Sandor Acs, Attila Kertesz, Gabor Kecskemeti
MTA SZTAKI, P.O. Box 63
1518 Budapest, Hungary
{*acs,attila.kertesz,kecskemeti*}*@sztaki.hu*

***Abstract*—Cloud computing represents a promising computing paradigm where computing resources have to be allocated to software for their execution. Self-manageable Cloud infrastructures are required to achieve that level of flexibility on one hand, and to comply to users' requirements specified by means of Service Level Agreements (SLAs) on the other. Such infrastructures should automatically respond to changing component, workload, and environmental conditions minimizing user interactions with the system and preventing violations of agreed SLAs. However, identification of sources responsible for the possible SLA violation and the decision about the reactive actions necessary to prevent SLA violation is far from trivial. First, in this paper we present a novel approach for mapping low-level resource metrics to SLA parameters necessary for the identification of failure sources. Second, we devise a layered Cloud architecture for the bottom-up propagation of failures to the layer, which can react to sensed SLA violation threats. Moreover, we present a communication model for the propagation of SLA violation threats to the appropriate layer of the Cloud infrastructure, which includes negotiators, brokers, and automatic service deployer.**

***Keywords*-Cloud Computing; SLA management; autonomic computing;**

I. Introduction

Cloud computing can be defined as the convergence and evolution of several concepts from virtualization, distributed application design, Grid and enterprise IT management to enable a more flexible approach for deploying and scaling applications [3], [19], [18]. Service provisioning in the Cloud is based on Service Level Agreements (SLAs) representing a contract signed between the customer and the service provider including the non-functional requirements of the service specified as Quality of Service (QoS). SLA considers obligations, service pricing, and penalties in case of agreement violations.

Flexible and reliable management of SLA agreements is of paramount importance for both, Cloud providers and consumers. On one hand, preventions of SLA violations ahead of time can avoid unnecessary penalties a provider has to pay in case of violations. Sometimes, simple actions like migrating VMs to available nodes can prevent SLA violations. On the other hand, based on flexible and timely reactions to possible SLA violations, interactions with the users can be minimized increasing the chance for Cloud computing to take roots as a flexible and reliable form of on demand computing.

However, current Cloud infrastructures lack appropriate mechanisms for the self-management of SLAs. Large body of work concentrates on monitoring of resource metrics of Cloud resources, which however cannot be easily mapped to SLA parameters [1], [2]. There is also considerable body of work done in the area of SLA management in general, which however is not related to Cloud infrastructures [15]. Thus, very little work has been done on identifications of SLA violations ahead of time, before they happen. Furthermore, there is a lack of appropriate mechanisms to identify which components of the Cloud infrastructure have to react in order to avert SLA violations.

In this paper we present *LAYSI - A Layered Approach for Prevention of SLA-Violations in Self-manageable Cloud Infrastructures*, which is embedded into the *FoSII project (Foundations of Self-governing ICT Infrastructures)* [8], an ongoing research project developing self-adaptable Cloud services. The *LAYSI* framework represents one of the building blocks of the *FoSII* infrastructure facilitating future SLA violation detection and propagation of the reactive actions to the appropriate layer of the Cloud infrastructure. We discuss a layered Cloud architecture utilizing hierarchically and loosely coupled components like negotiator, broker or automatic service deployer. For the decision making we use knowledge databases proposing reactive actions by utilizing case based reasoning - a process of solving problems based on past experience. Based on the novel communication model we present how possible SLA violations can be identified and propagated to the layer of the Cloud infrastructure, which can execute appropriate reactive actions in order to advert SLA violations.

The main contributions of this paper are: (i) discussion on the solution for mapping low-level resource metrics to SLA parameters; (ii) description of the integrated SLA-aware Cloud architecture suitable for the propagation of the SLA violation threats; (iii) concept for the realization

of the knowledge database using case based reasoning; (iv) architecture for the autonomic management and propagation of SLA violation threats.

The rest of this paper is organized as follows: Section II presents the related work. In Section III we present the architecture for the autonomic management of Cloud services and the approach for mapping low-level resource metrics to SLA parameters. In Section IV we discuss the LAYSI architecture. In particular we discuss the concept of knowledge databases and the SLA manager responsible for the autonomic management of SLA violation threats. Section V presents our conclusions and describes the future work.

II. Related Work

We classify related work into (i) monitoring of Cloud/Grid/Web services [1], [2]; (ii) SLA management including QoS management [9], [6], [14]; (iii) and self-management of Cloud/Grid/SOA services [15]. Since there is very little work on monitoring, SLA management, and self-mamagement in Cloud systems we look particularly into related areas, i.e., Grid and SOA based systems.

GridRM is an open-source project trying to provide a unified way of accessing different monitored data sources. Every domain needs a Java-based gateway to collect and normalize events from the local monitoring system. However, it does not provide mapping of monitored values to SLA parameters [1].

Frutos et al. [9] discuss the main approach of the EU project BREIN [6]: to develop a framework, which extends the characteristics of computational Grids by driving their usage into new target areas in the business domain. BREIN deals with the provision of the basic infrastructure these new business models need: enterprise system interoperability, flexible relationships, dynamicity in business processes, security mechanisms, and enhanced SLA and contract management. However, BREIN applies SLA management to Grids, whereas we target SLA management in Clouds. Koller et al. [14] discuss autonomous QoS management using a proxy-like approach. The implementation is based on WS-Agreement. Thereby, SLAs can be exploited to define certain QoS parameters that a service has to maintain during its interaction with a specific customer. However, their approach is limited to Web services and does not consider requirements of Cloud Computing infrastructures like scalability.

Based on the defined workflow adaptations as MAPE[1] decision making [15], Lee et al. discuss the application of autonomic computing to the adaptive management of Grid workflows.

III. FoSII Infrastructure

In this section we present an overview of the *FoSII* infrastructure and its relation to the *LAYSI* framework. In particular we describe the mapping of low level metrics to high level SLAs. Thereafter, we discuss the SLA-based layered Cloud infrastructure.

[1]Monitoring, Analysis, Planning, Execution

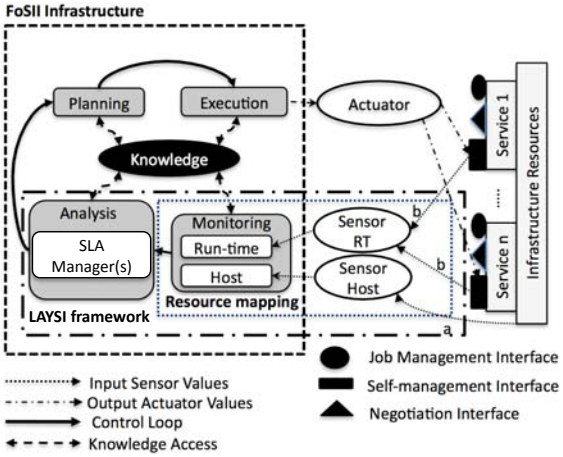


Figure 1. FoSII infrastructure

A. FoSII overview

The FoSII infrastructure is used to manage self-adaptable Cloud services following the MAPE lifecycle. Each FoSII service implements three interfaces: (i) negotiation interface necessary for the establishment of SLA agreements, (ii) job-management interface necessary to start the job, upload data, and similar job management actions, and (iii) self-management interface necessary to devise actions in order to prevent SLA violations.

The self-management interface shown in Figure 1 is implemented by each Cloud service and specifies operations for sensing changes of the desired state and for reacting to those changes. The host monitor sensors continuously monitor the infrastructure resource metrics (input sensor values arrow *a* in Figure 1) and provide the autonomic manager with the current resource status. The run-time monitor sensors sense future SLA violation threats (input sensor values arrow *b* in Figure 1) based on resource usage experiences and predefined threat thresholds. The mapping between the sensed host values and the values of the SLA parameters is described next.

B. Mapping of Low level Metrics to High-level SLAs

In order to explain our mapping approach we consider the Service Level Objectives (SLOs) as shown in Table I including incoming bandwidth, outgoing bandwidth, storage, and availability.

As shown in Figure 1 we distinguish between *host monitor* and *runtime monitor*. Resources are monitored by the *host monitor* using arbitrary monitoring tools (e.g. Ganglia [17]). Resource metrics include, e.g., down-time, up-time, available storage. Based on the predefined mappings stored in a database, monitored metrics are periodically mapped to

SLA Parameter	Value
Incoming Bandwidth (IB)	$>$ 10 Mbit/s
Outgoing Bandwidth (OB)	$>$ 12 Mbit/s
Storage (St)	$>$ 1024 GB
Availability (Av)	$\geq$ 99%

Table I
SAMPLE SLA PARAMETER OBJECTIVES

the SLA parameters. An example SLA parameter is service availability Av, (as shown in Table I), which is calculated using the resource metrics $downtime$ and $uptime$ and the mapping rule looks like the following:

$Av = (1 - downtime/uptime) * 100$

The mapping rules are defined by the provider using appropriate Domain Specific Languages (DSLs). These rules are used to compose, aggregate, or convert the low-level metrics to form the high-level SLA parameter including mappings at different complexity levels, e.g., $1 : n$ or $n : m$. The concept of detecting future SLA violation threats is designed by defining a more restrictive threshold than the SLA violation threshold known as threat threshold. Thus, calculated SLA values are compared with the predefined threat threshold in order to react before SLA violations happen. The generation of *threat thresholds* is far from trivial and is part of our ongoing work including sophisticated methods for the system state management as described in Section IV-A.

As described in [7] we implemented a highly scalable framework for mapping Low Level Resource Metrics to High Level SLA Parameters (LoM2HiS framework) facilitating the exchange of large numbers of messages. We designed and implemented a communication model based on the Java Messaging Service (JMS) API, which is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients. We use Apache ActiveMQ as a JMS provider that can manage the sessions and queues.

Once possible SLA violation threats are detected, reactive actions are taken in order to prevent real SLA violations. In the following we discuss the layered Cloud architecture followed by the discussion of the novel concept for the SLA violation threat propagation.

C. SLA-based Layered Cloud Infrastructures

In the following we present a unified service architecture that builds on three main areas [11]: agreement negotiation, brokering, and service deployment using virtualization. We suppose that service providers and service consumers meet on demand and usually do not know about the negotiation protocols, document languages or required infrastructure of the potential partners. The architectures' components are loosely coupled using SLAs between the components. Thus, in case of failures components can be exchanged easily by renegotiating with another instance, e.g. another broker.

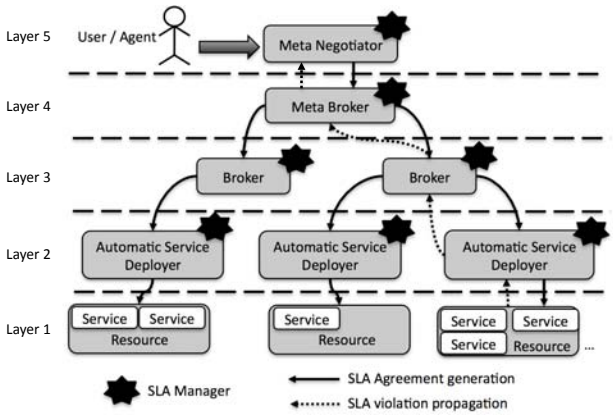


Figure 2. LAYSI infrastructure

Figure 2 shows our proposed general architecture. In the following we discuss the actors of the proposed architecture:

- **User:** A person, who wants to use a service, an agent or software application acting on behalf of a user.
- **Meta Negotiator:** A component that manages SLAs. It mediates between the user and the meta-broker, selects services, and resources considering prescribed protocols, negotiation strategies, and security restrictions as described in [5].
- **Meta Broker:** Its role is to select a broker that is capable of deploying a service with the specified user requirements as described in [12].
- **Broker:** It interacts with virtual or physical resources, and in case the required service needs to be deployed it interacts directly with the Automatic Service Deployer (ADS) [13].
- **Automatic Service Deployer:** It installs the required service on the selected resource on demand as described in [10].
- **Service:** The service that users want to deploy and/or execute is described using the concept of *virtual appliances*.
- **Resource:** Physical machines, network, or storage elements on which virtual machines can be deployed/installed.

The SLA negotiation is done as following: The *User* starts a negotiation for executing a service with certain QoS requirements. Then, the *Meta negotiator* asks the *Meta broker*, if it could execute the service with the specified requirements including required negotiation or security protocols. The *Meta broker* matches the requirements to the properties of the available *Brokers* and replies with an acceptance or a different offer for renegotiation. The aforementioned steps may continue for renegotiations until both sides agree on the terms (to be written to an SLA document) following the specific negotiation strategy or auction. Thereafter, the *User* calls the service with the *Service Description (SD)* and

the agreed *SLA*. SDs describe a master image by means of a self-contained software stack (OS, middleware, applications, data, and configuration) that fully captures the functionality of the component type. Moreover, the SD contains information and rules necessary to automatically create service instances from a single parametrized master.

Meta-negotiator passes the SD and the possibly transformed SLA (using a protocol the selected broker understands) to the Meta broker. The meta broker calls the selected Broker with the SLA and a possibly translated SD (to the language of the Broker). The Broker executes the service with respect to the terms of the SLA. The ASD monitors the states of the virtual resources and deploys services, as already stated in Figure 1. As shown in Figure 2 SLA generation is done top-down as already described. Management of the SLA threat violation is done bottom-up on behalf of the *SLA manager*, which is implemented by each component of the SLA layered architecture.

Table II shows the implementation choices for the layered Cloud architecture and the possible reactive actions each layer can perform. We use *Meta Negotiator* [4], *Meta Broker* [12], *GTBroker* [13] and *Automatic Service Deployer* [10] components, which have already been used and evaluated to build an SLA-based resource virtualization environment for on-demand service provision [11].

IV. LAYSI: A LAYERED APPROACH FOR SLA-VIOLATION PROPAGATION

In the following we present an architecture for the propagation of the sensed critical SLAs, which might be violated in the future. In particular we focus on two components: the *knowlege database* providing reactive action for possible detected SLA violation threats considering SLA threat thresholds and the current system status (Section IV-A), and the *SLA manager* propagating the sensed SLA violation threats to the appropriate layer of the infrastructure for preventive actions (Section IV-B).

A. Knowlegde DBs

For the decision making we use knowledge databases proposing the reactive actions by utilizing case based reasoning. *Case Based Reasoning (CBR)* is the process of solving problems based on past experience. It tries to solve a *case* (a formatted instance of a problem) by looking for similar cases from the past and reusing the solutions of these cases to solve the current one. In general, a typical CBR cycle consists of the following phases assuming that a new case has just been received: (i) retrieve the most similar case or cases to the new one, (ii) reuse the information and knowledge in the similar case(s) to solve the problem, (iii) revise the proposed solution, (iv) retain the parts of this experience likely to be useful for future problem solving.

As shown in Figure 3, a complete case consists of (a) the ID of the application being concerned (line 2, Figure 3); (b) the initial case measured by the monitoring component

```
1.  (
2.   (App,  1),
3.    (
4.    ((Incoming Bandwidth, 12.0),
5.     (Outgoing Bandwidth, 20.0),
6.     (Storage, 1200),
7.     (Availability, 99.5),
8.     (Running on PMs, 1)),
9.     (Physical Machines, 20)
10.   ),
11.  "Increase Incoming Bandwidth share by 5%",
12.  (
13.   ((Incoming Bandwidth, 12.6),
14.    (Outgoing Bandwidth, 20.1),
15.    (Storage, 1198),
16.    (Availability, 99.5),
17.    (Running on PMs, 1)),
18.    (Physical Machines, 20)
19.   ),
20.  0.002
21. )
```

Figure 3. CBR example

and mapped to the SLAs consisting of the SLA parameter values of the application and global Cloud information like number of running virtual machines (lines 4-10, Figure 3); (c) the executed action (line 11, Figure 3); (d) the resulting case measured some time interval later (lines 12-18, Figure 3) as in (b); and (e) the resulting utility (line 20, Figure 3).

We distinguish between two working modes of the knowledge DB: *active* and *passive*. In the active mode system states and SLA values are periodically stored into the DB. Thus, based on the observed violations and correlated systems states, cases are obtained as input for the knowledge DB. Furthermore, based on the utility functions, we evaluate the quality of the reactive actions and generate threat thresholds. In the passive mode notification are sent by the SLA manager (or LoM2HiS framework in case the layer=1) as described in Section III-B.

However, the output of the DB does not tell anything about how to react to the proposed actions as for example the suggested action *Increase Incoming Bandwidth share by 5%* depicted in Figure 3. An obvious reaction would be to increase the bandwidth share by the particular resource. However, if this is not possible due to resource restriction, current load, and services with competing priorities, the suggested action has to be propagated to the next layer. Then, in the next layer ASD could migrate the virtual appliance as specified in Table II (reactive actions of ASD: suspend, shut-down, and migrate VAs). This propagation can be continued until a specific layer is able to react to the particular suggested action.

In the following we discuss how the SLA manager can propagate the desired changes to the particular layer of the infrastructure, which can take appropriate actions.

B. SLA Manager

The SLA manager implements the component's self-management interfaces and invokes the self-management

Layer	Sample Implementation	Actions
Meta Negotiator	Meta Negotiator in Brandic et al. [4]	start new meta-negotiation
Meta Broker	Meta-Broker in Kertesz et al. [12]	allocate new broker
Broker	GTBroker in Kertesz et. al. [13]	start, stop, and suspend ASD instances
Automatic Service Deployment (ASD)	ASD in Kecskemeti et al. [10]	suspend, shut-down, and migrate virtual appliances (VAs)

Table II
IMPLEMENTATION CHOICES AND THE POSSIBLE REACTIVE ACTIONS OF THE PARTICULAR LAYER

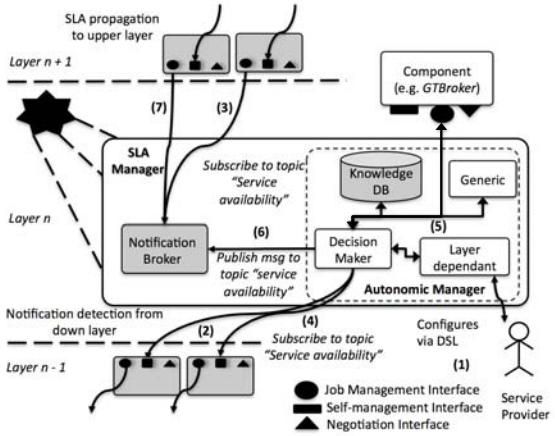


Figure 4. SLA Manager

interface of the upper layer in case the announced SLA violation threat cannot be solved by the layer's SLA manager. The SLA manager considers two main parts: the *Notification Broker* implemented using the WS-Notification mechanism and the *Autonomic Manager* managing the access to the knowledge DB, accessing the job management interfaces of the component, and making the decision whether the SLA violation threats can be handled by the layer or not.

Autonomic Manager: The Autonomic Manager receives notifications from the lower layer or from the *LoM2HiS framework* in case the layer=1. Thereafter, the knowledge DB is accessed in order to receive states, which should be achieved. The decision maker consists of two parts: the layer independent part managing the DB access and notifications, i.e., the *generic part* and the *layer dependent* part, which implements access to the components' interfaces e.g., in order to take reactive actions. The user can customize the autonomic manager, e.g., taking into account the job management interface of the component by modifying the component's dependent part using Domain Specific Languages (DSLs) (step 1, Figure 4). Customization could include for example utilization of the reactive actions of a component as shown in Table II. The notification mechanism for the propagation of SLA violation threats is explained next.

Notification Broker: The *SLA manager* employs the *WS-Notification* mechanism [21], which provides a set of standard interfaces to use the notification design pattern with services. WS-Notification is defined by three specifications: (i) *WS-Topics*; (ii) *WS-BaseNotification*; and (iii) *WS-BrokeredNotification*. The *WS-Topics* present a set of items of interest for subscription. Topics are very versatile and highly customizable. They even allow us to create topic trees, where a topic can have a set of child topics. *WS-BaseNotification* defines the standard interfaces used by the notification producer and consumer. The *WS-BrokeredNotification* delivers notification from the producer to the consumer through an intermediate entity (broker). The SLA manager is also equipped with a queueing network. The queues are used to temporarily store the notifications for processing. With the queueing networks, there is the possibility of selectively processing higher priority notifications against the lower priority ones.

Decision makers can subscribe different topics as shown in Figure 4, step 2 and 3. Once the SLA violation threat is detected the autonomic manager tries to find a reactive action by accessing the DB utilizing case based reasoning (step 5, Figure 4). The decision components decides whether the SLA violation threats can be deferred. If the SLA violation threats cannot be deferred at that certain layer, the SLA violation threats are propagated by publishing a message to the specific topic, e.g., to topic *service availability* as shown in step 6, Figure 4. Thereafter, all listeners (i.e., components of the layer $n+1$) are notified, step 7, Figure 4. The topics are organized in a hierarchical way considering the learning function of the CBR database. Based on the observed sensed violations, reactive actions, and the utility of the reactive action we define dependencies of the reactive actions which can be reflected in the topic hierarchy. The development of the advanced techniques for the automatic definition and utilizations of the topics hierarchies is subject of our ongoing work.

The LoM2HiS framework publishes monitored SLA parameters as a specific message of a WS-Topic. Thereafter, preventive actions of the SLA violation threats should be notified and handled at the ASD layer. In case the SLA violation threats can not be handled at layer n, the SLA manager publishes the problem to layer $n+1$. In the worst case, this propagation continues to the top level, i.e., the Meta Negotiator, which informs the user about the problem for a possible renegotiation or aborting the service execution.

V. Conclusion and Future Work

In this paper we presented how possible and costly SLA violations can be prevented by utilizing a layered SLA based Cloud infrastructure. Based on the novel approach for mapping low-level resource metrics to SLA parameters we can identify possible SLA violations. We devised a layered Cloud architecture for the bottom-up propagation of failures to the layer, which can react to sensed SLA violation threats. Moreover, we presented a communication model for the propagation of SLA violation threats to the appropriate layer of the Cloud infrastructure, which includes meta-negotiators, brokers, and automatic service deployer.

In the future we will integrate learning functions into the CBR databases in order to identify whether the propagation had a positive impact on the prevention of SLA violations. We plan to integrate trade-off analysis to examine how costly the interventions for the possible future SLA violations are instead of just reacting to occurred violations.

Acknowledgment

The work described in this paper was partially supported by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII), and by the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

[1] M. Baker and G. Smith. *GridRM: A resource monitoring architecture for the grid. Lecture Notes in Computer Science*, Vol. 2536, pp. 268-273, 2002

[2] W. Fu and Q. Huang. *GridEye: A service-oriented grid monitoring system with improved forecasting algorithm.* GCCW'06, pp. 5-12, 2006.

[3] R. Buyya, C.S Yeo, S. Venugopal, J.Broberg, and I. Brandic. *Cloud computing and emerging IT platforms: Vision Hype, and Reality for delivering computing as the 5th utility.* Future Generation Computer Systems Vol. 25(6) pp. 599-616, June 2009.

[4] I. Brandic, D. Music, S. Dustdar. *Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services.* GMAC09 in conjunction with ICAC09, Barcelona, Spain, June 15-19, 2009.

[5] I. Brandic, S. Venugopal, M. Mattess, and R. Buyya.*Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services*. SENOPT08, in conjunction with International Conference on High Performance Computing 2008 (HiPC 2008), Bangalore, India, December 17 - 20, 2008.

[6] Brein Project (Business objective driven reliable and intelligent Grids for real business), http://www.eu-brein.com 2009

[7] V. C. Emeakaroha, I.Brandic, M. Maurer, S. Dustdar. *Low Level Metrics to High Level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in Cloud environments.* The 2010 High Performance Computing and Simulation Conference (HPCS 2010) June 28-July 2, 2010, Caen, France.

[8] Foundations of Self-governing ICT Infrastructures (FoSII), http://www.infosys.tuwien.ac.at/linksites/FOSII

[9] H.M. Frutos and I. Kotsiopoulos. *BREIN: Business Objective Driven Reliable and Intelligent Grids for Real Business*. International Journal of Interoperability in Business Information Systems, 3(1) 2009.

[10] G. Kecskemeti, P. Kacsuk, G. Terstyanszky, T. Kiss, T. Delaitre. *Automatic Service Deployment Using Virtualisation.* 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), Toulouse, France, 13-15 February 2008.

[11] A. Kertesz, G. Kecskemeti, I. Brandic. *An SLA-based Resource Virtualization Approach for On-demand Service Provision.* VTDC 2009, In conjunction with ICAC09, Barcelona, Spain, June 15-19, 2009.

[12] A. Kertesz and P. Kacsuk. *GMBS: A New Middleware Service for Making Grids Interoperable.* Future Generation Computer Systems, Volume 26, Issue 4, April 2010, Pages 542-553.

[13] A. Kertesz, G. Sipos, P. Kacsuk. *Multi-Grid Brokering with the P-GRADE Portal.* In Post-Proceedings of the Austrian Grid Symposium (AGS'06), pp. 166-178, OCG Verlag, Austria, 2007.

[14] B. Koller, L. Schubert. *Towards autonomous SLA management using a proxy-like approach.* Multiagent Grid Syst. Vol.3, 2007.

[15] K. Lee, R. Sakellariou, N. W. Paton, A. A. A. Fernandes. Workflow *Adaptation as an Autonomic Computing Problem*. WORKS'07 pages 29-34. in conjunction with HPDC 2007, Monterey, California, USA, 2007.

[16] G. Lin, G. Dasmalchi, J. Zhu. *Cloud Computing and IT as a Service: Opportunities and Challenges*. IEEE International Conference on Web Services (ICWS08), Beijing China, 23-26 Sept. 2008.

[17] M.L Massie, B.N Chun and D.E Culler. *Ganglia distributed monitoring system: design implementation, and experience.* Parallel Computing, Vol. 30 pp. 817-840, 2004.

[18] D. Nurmi, R. Wolski, Ch. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov. *The Eucalyptus Open-source Cloud-computing System*. Proceedings of Cloud Computing and Its Applications 2008, Chicago, Illinois, October 2008.

[19] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. B.-Y., W. Emmerich, F. Galan. *The RESERVOIR Model and Architecture for Open Federated Cloud Computing.*, IBM Journal of Research and Development, 53(4) (2009)

[20] R. Wolski, N.T. Spring and J. Hayes. *The network weather service: A distributed resource performance forecasting service for metacomputing.* In Journal of Future Generation Computing Systems, Vol. 15, pp. 757-768, 1999.

[21] WS-Notification, http://www.ibm.com/developerworks/webservices/library/specification/ws-notification

A Soft-Constraint Based Approach to QoS-Aware Service Selection *

Mohamed Anis Zemni[1], Salima Benbernou[1], Manuel Carro[2]

[1] LIPADE, Université Paris Descartes, France
[2] Facultad de Informática, Universidad Politécnica de Madrid, Spain
mohamedaniszemni@gmail.com, salima.benbenrou@paridescartes.fr, mcarro@fi.upm.es

Abstract. Service-based systems should be able to dynamically seek replacements for faulty or underperforming services, thus performing self-healing. It may however be the case that available services do not match all requirements, leading the system to grind to a halt. In similar situations it would be better to choose alternative candidates which, while not fulfilling all the constraints, allow the system to proceed. *Soft constraints*, instead of the traditional *crisp* constraints, can help naturally model and solve replacement problems of this sort. In this work we apply soft constraints to model SLAs and to decide how to rebuild compositions which may not satisfy all the requirements, in order not to completely stop running systems.
Keywords: Service Level Agreement, Soft Constraints.

1 Introduction

A (web) service can be defined as a remotely accessible software implementation of a resource, identified by a URL. A set of protocols and standards, such as WSDL, facilitate invocation and information exchange in heterogeneous environments. Software services expose not only functional characteristics, but also non-functional attributes describing their Quality of Service (QoS) such as availability, reputation, etc. Due to the increasing agreement on the implementation and management of the functional aspects of services, interest is shifting towards non-functional attributes describing the QoS. Establishing QoS contracts, described in the Service Level Agreement (SLA), that can be monitored at runtime, is therefore of paramount importance. Various techniques [1] to select services fulfilling functional and non-functional requirements have been explored, some of them based on expressing these requirements as a constraint solving problem [2, 3] (CSP). Traditional CSPs can either be fully solved (when all requirements are satisfied) or not solved at all (some requirements cannot be satisfied). In real-life cases, however, over-constraining is common (e.g., because available services offer a quality below that required by the composition), and problems are likely not to have a classical, crisp solution. Solving techniques for *soft* CSPs (SCSP) [4–6] can generate solutions for overconstrained problems by allowing some constraints to remain unsatisfied.

* The research leading to these results has received funds from the European Community's Seventh Framework Programme FP7/2007-20013 under grant agreement 215483 (S-CUBE). Manuel Carro was also partially supported by Spanish MEC project 2008-05624/TIN *DOVES* and CM project P2009/TIC/1465 (*PROMETIDOS*).

A C-semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ s.t.

- A is a set and $\mathbf{0} \in A$, $\mathbf{1} \in A$.
- $\sum$ (the *additive* operation)[a] is defined on subsets of A as follows:
  - $+$ is commutative ($a + b = b + a$), associative ($a + (b + c) = (a + b) + c$), with unit element $\mathbf{0}$ ($a + \mathbf{0} = a$) and absorbing element $\mathbf{1}$ ($a + \mathbf{1} = \mathbf{1}$).
  - $\sum \emptyset = \mathbf{0}$ and for all $a \in A$, $\sum\{a\} = a$.
  - Given any set of indices S, $\sum_{i \in S}(\bigcup A_i) = \sum(\{\sum_{i \in S} A_i\})$ (flattening).
- $\times$ (the *multiplicative* operation) is associative, commutative, $a \times \mathbf{1} = a$ and $a \times \mathbf{0} = \mathbf{0}$.
- $\times$ distributes over $+$, i.e., $a \times (b + c) = (a \times b) + (a \times c)$.

[a] Written as infix $+$ when applied to a two-element set.

Fig. 1: Definition of a C-Semiring for Soft Constraints.

Our framework takes into consideration the penalties agreed upon on the SLA by building a new (Soft) Service Level Agreement (SSLA) based on preferences where strict customer requirements are replaced by soft requirements allowing a suitable composition. This agreement has to include penalty terms to be applied while the contract terms are violated.

2 Soft Constraints in a Nutshell

A CSP defines a set of variables whose ranges we assume a finite domain (FD)[3] and a set of constraints which restrict the values these variables can take. A solution for a CSP is an assignment of a value to *every* variable s.t. all the constraints are simultaneously satisfied. Soft constraints [5, 6] generalize classical CSPs by adding a preference level to every tuple in the domain of the constraint variables. This level can be used to obtain a suitable solution which may not fulfill all constraints, which optimizes some metrics , and which in our case will be naturally applied to the requirements of the users.

The basic operations on soft constraints (building a constraint conjunctions and projecting on variables) need to handle preferences in a homogeneous way. This requires the underlying mathematical structure of classical CSPs to change from a cylindrical algebra to a semiring algebra, enriched with additional properties and termed a *C-semiring* (Figure 1). In it, A provides the levels of preference of the solutions and it can be proved that it is a lattice with partial order $a \preceq b$ iff $a + b = b$, minimum $\mathbf{0}$, and maximum $\mathbf{1}$. When solutions are combined or compared, preferences are accordingly managed using the operations $\times$ and $+$. Note that the theory makes no assumptions as to what the preferences mean, or how they are actually handled: $\times$ and $+$ are placeholders for concrete definitions which can give rise to different constraint systems, such as fuzzy constraints, traditional constraints, etc.

Figure 2 summarizes some basic definitions regarding soft constraints. A constraint takes a tuple of variables and assigns it a tuple of concrete values in the domain of the

[3] CSPs can be defined on infinite domains, but assume a FD here because it can accommodate many real-life problems, as witnessed by the relevance of FD in industrial applications, and because soft constraint theory requires finiteness.

Definition 1 (Constraint). *Given a c-semiring $\langle A, +, \times, 0, 1\rangle$, a set of variables V, and a set of domains D, one for every variable in V, a constraint is the pair $\langle def, con\rangle$ where $con \subseteq V$ and $def : D^{|con|} \rightarrow A$.*

Definition 2 (Soft Constraint Satisfaction Problem SCSP). *A SCSP is a pair $\langle C, con\rangle$ where $con \subseteq V$ and C is a set of constraints. C may contain variables which are not in con, i.e., they are not interesting for the final result. In this case the constraints in C have to be projected onto the variables in con.*

Definition 3 (Constraint combination). *Two constraints $c_1\langle def_1, con_1\rangle$ and $c_2 = \langle def_2, con_2\rangle$ can be combined in $c_1 \otimes c_2 = \langle def, con\rangle$ by taking all the variables in the original constraints ($con = con_1 \bigcup con_2$) and assigning to every tuple in the new constraint a preference value which comes from combining the values in the original constraints: $def(t) = def_1(t \downarrow^{con}_{con_1}) \times def_2(t \downarrow^{con}_{con_2})$.*

Definition 4 (Projection). *Given a soft constraint $c = \langle def, con\rangle$ and a set of variables $I \subseteq V$, the projection of c over I, denoted by $c \Downarrow_I$ is the constraint $\langle def', con'\rangle$ where $con' = con \bigcap I$ and $def'(t') = \sum_{\{t | t\downarrow^{con}_{con\cap I} = t'\}} def(t)$.*

Definition 5 (Solution). *A solution of a SCSP problem $\langle C, con\rangle$ is the constraint $(\otimes C) \Downarrow_{con}$, i.e., the combination (conjunction) of all the constraints in C projected over all the variables con of interest.*

Fig. 2: Definitions for Soft Constraints.

variables, plus a preference value (belonging to the set A). Constraints can be combined into other constraints (with $\otimes$, similar to conjunction) and projected ($\Downarrow^X_Y$) onto a tuple of variables. The preference value of every tuple in a constraint conjunction is worked out by applying $\times$ to the preference values of the tuples in the individual constraints. Projections eliminate "columns" from tuples and retain only the non-removed tuple components. Repeated rows may then appear, but only one is retained, and its preference is calculated applying $+$ to the preferences of the repeated tuples. Since a solution is a projection on some selected set of variables, the preferences of solutions are naturally calculated using the projection operation. Usually the tuple with the highest preference value is selected as the "optimal" solution.

3 Soft Service Level Agreement and SCSPs

A Service Level Agreement (SLA) [7] is a contract between provider(s) and client(s) specifying the guarantees of a service, the expected quality level, and the penalties to be applied in case of unfulfillment of duties, and it is therefore an important quality management artifact. The SLA can be used to identify the responsible of a malfunction and to decide which action (if any) has to be taken. An SLA should, therefore, be realistic, achievable, and maintainable.

An SLA has a rich structure from which we underline the properties of the services, including those measurable aimed at expressing guarantees. This part provides a set Υ of variables υ_i (whose meaning is explained in the service description) and their domains $\delta_i \in \Delta$, which can be established by the metric attribute. A Soft SLA (SSLA) is similar to a SLA but with the addition of a set of user preferences and of penalties

Definition 6 (Preference). *The set* $Pr = \{\langle \delta_i, \upsilon_i, a_i \rangle | \delta_i \in \Delta, \upsilon_i \in \Upsilon, a_i \in A\}$ *where* δ_i *is the sub-domain that the ith preference belongs to,* υ_i *is the variable defining the preferences, and* a_i *is semiring value, representing the preferences in an SSLA.*

Definition 7 (Penalty). *The set* $Pn = \{pn_i \mid \exists pr_i \; s.t. \; \upsilon_i \notin \delta_i\}$ *represents the penalties.*

Definition 8 (SSLA document). *A SSLA document is a tuple* $\zeta = \langle \Upsilon, \Delta, A, Pr, Pn, T \rangle$ *where* Υ *is a set of variables* υ_i*,* Δ *is a set of variable domains* δ_i *(one for each variable), Pr is a set of preferences* Pr_i*,* Pn *is a set of penalties* Pn_i *to apply when the preferences are not satisfied and* T *is a set of pairs* $\langle pr_i, pn_i \rangle$ *which associates preferences with the penalties to apply in case of violation.*

Fig. 3: Definitions related to a soft SLA.

associated to contract breaking (respectively, Pr and Pn). The preferences are used to make a composition in the presence of unsatisfied requirements and the penalties are used to refine found solutions and to protect each party from the violation of the contract terms. These notions are depicted in Figure 3. The i-th penalty $pn_i \in Pn$ is applied when the i-th preference $pr_i \in Pr$ is not satisfied.

3.1 Extending SCSP Using Penalties

We will adapt the SCSP framework to handle explicitly penalties for service selection and to build a Soft Service Level Agreement including preferences and penalties. In this framework, service selection has three phases:

1. Model the characteristics for the selection using soft constraints.
2. Assuming a pre-selection is made using functional requirements, rank candidate services using non-functional requirements and the constraint preferences.
3. We assign penalties to unmet user preferences, and these penalties are used to rank solutions having the same constraint preferences.

Figure 4 shows the definitions for this extended SCSP framework. We extend the application of semiring operations to penalties. Variables are assumed to take values over subdomains which discretize a continuous domain, and which for brevity we represent using identifiers in $D_{\{\}}$. The constraint preference function *def* is also adapted in order to apply it both to preferences and to penalties. The projection operation is kept as in the SCSP framework.

3.2 An Example

A delivery service has an order-tracking web service. Companies wishing to hire this service want to have in the contract non-functional criteria such as availability, reputation, response time and cost.

Phase 1 Let $CS = \langle S_p, D_{\{\}}, V \rangle$ be a constraint system and $P = \langle C, con \rangle$ be the problem to be solved, where $V = con = \{\underline{\text{A}}\text{vailability}, \underline{\text{R}}\text{eputation, response } \underline{\text{T}}\text{ime, co}\underline{\text{St}}\}$, $D_{\{\}} = \{\{a_1, a_2\}, \{r_1, r_2\}, \{t_1, t_2, t_3\}, \{s_1, s_2\}\}$, $S_p = \langle [0, 1], Pn, \max, \min, 0, 1 \rangle$, $C = \{c_1, c_2, c_3, c_4\}$. For simplicity, variables and their domains have been written in the same order.

Definition 9 (CP-semiring). *A CP-semiring is a tuple $S = \langle A, Pn, +, \times, 0, 1\rangle$, extending a C-semiring. A and Pn are two sets with lattice structure stating preference values for solutions and penalties. Operations $\times$ and $+$ are applied when constraints are combined or projected.*

Definition 10 (Constraint System). *A constraint system is a tuple $CS = \langle S, D_{\{\}}, V\rangle$, where S is c-semiring, $D_{\{\}}$ represents the set of identifiers of subdomains, and V is the ordered set of variables.*

Definition 11 (Constraint). *Given a constraint system $CS = \langle S_p, D_{\{\}}, V\rangle$ and a problem $P = \langle C, con\rangle$, a constraint is the tuple $c = \langle def_c, type\rangle$, where $type$ represents the type of constraint and def_c is the definition function of the constraint, which returns the tuple*

$$def : D^{|con|} \rightarrow \langle p_r, p_n\rangle,$$

Definition 12 (Soft Constraint Satisfaction Problem SCSP). *Given a constraint system $CS = \langle S, D_{\{\}}, V\rangle$, an SCSP over CS is a pair $P = \langle C, con\rangle$, where con, called set of variables of interest for C, is a subset of V and C is a finite set of constraints, which may contain some constraints defined on variables not in con.*

Fig. 4: CP-Semiring.

A set of penalties, ranked from the most to then less important one, has been set: $pn_i \preceq pn_j$ if $i \leq j$. The above shown values of the variable domains comes from a discretization such as $availability \in \{[0, 0.5[, [0.5, 1]\}$, $reputation \in \{[0, 0.6[, [0.6, 1]\}$, $response\ time \in \{[20, \infty[, [5, 20[, [0, 5[\}$, $cost \in \{[1000, 1500[, [1500, 3000]\}$.

Let us consider the following constraints: $c_1 = \langle def_{c1}, \{\text{availability, reputation}\}\rangle$, $c_2 = \langle def_{c2}, \{\text{response time}\}\rangle$, $c_3 = \langle def_{c3}, \{\text{availability, reputation, cost}\}\rangle$, $c_4 = \langle def_{c4}, \{\text{reputation, response time}\}\rangle$, where the preference values and corresponding penalties are in Table 1. For example, for the tuple $\langle a_2, r_1\rangle$, attributes "availability" and "reputation" are respectively assigned subdomains $[0.5, 1]$ and $[0, 0.6[$. The function $def_{c1}(\langle a_2, r_1\rangle) = \langle 0.5, pn_3\rangle$ shows that these attribute values have a preference 0.5 and company is ready to sign away this preference for a penalty pn_3.

Phase 2 Given the model, we define constraint combination to keep the minimum value of preferences (resp. for the penalties). For example $def_{c1}(\langle a_2, r_1\rangle) \otimes def_{c2}(\langle t_3\rangle) = \min(\langle 0.5, pn_3\rangle, \langle 0.25, pn_6\rangle) = \langle 0.25, pn_3\rangle$ and so on with all the tuples to obtain $c_{1,2}$. Next, we would combine $c_{1,2}$ and c_3 to get $c_{1,2,3} = c_{1,2} \otimes c_3$ and so on, until all constraints have been combined. Table 2 shows the results of possible combinations.

$\langle A, R\rangle$	def_{c1}	$\langle T\rangle$	def_{c2}	$\langle A, R, S\rangle$	def_{c3}	$\langle R, T\rangle$	def_{c4}
$\langle a_1, r_1\rangle$	$\langle 0, -\rangle$	$\langle t_1\rangle$	$\langle 0.25, pn_6\rangle$	$\langle a_1, r_1, s_1\rangle$	$\langle 0.25, pn_8\rangle$	$\langle r_1, t_1\rangle$	$\langle 0.5, pn_6\rangle$
$\langle a_1, r_2\rangle$	$\langle 0.25, pn_1\rangle$	$\langle t_2\rangle$	$\langle 0.5, pn_5\rangle$	$\langle a_1, r_1, s_2\rangle$	$\langle 0.25, pn_1\rangle$	$\langle r_1, t_2\rangle$	$\langle 0.5, pn_5\rangle$
$\langle a_2, r_1\rangle$	$\langle 0.5, pn_3\rangle$	$\langle t_3\rangle$	$\langle 1, pn_7\rangle$	$\langle a_1, r_2, s_1\rangle$	$\langle 0.5, pn_1\rangle$	$\langle r_1, t_3\rangle$	$\langle 0, -\rangle$
$\langle a_2, r_2\rangle$	$\langle 0.75, pn_3\rangle$			$\langle a_1, r_2, s_2\rangle$	$\langle 0.25, pn_3\rangle$	$\langle r_2, t_1\rangle$	$\langle 0.75, pn_2\rangle$
				$\langle a_2, r_1, s_1\rangle$	$\langle 0.75, pn_9\rangle$	$\langle r_2, t_2\rangle$	$\langle 0.75, pn_4\rangle$
				$\langle a_2, r_1, s_2\rangle$	$\langle 0.5, pn_8\rangle$	$\langle r_2, t_3\rangle$	$\langle 1, pn_2\rangle$
				$\langle a_2, r_2, s_1\rangle$	$\langle 0.75, pn_2\rangle$		
				$\langle a_2, r_2, s_2\rangle$	$\langle 0.25, pn_1\rangle$		

Table 1: Constraint definitions.

$\langle A,R,T,S\rangle$	$\langle pr,pn\rangle$	$\langle A,R,T,S\rangle$	$\langle pr,pn\rangle$	$\langle A,R,T,S\rangle$	$\langle pr,pn\rangle$	$\langle A,R,T,S\rangle$	$\langle pr,pn\rangle$
$\langle 2,2,3,1\rangle$	$\langle 0.75,pn_2\rangle$	$\langle 2,2,1,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 1,2,1,1\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 1,1,3,2\rangle$	$\langle 0.0,-\rangle$
$\langle 2,2,2,1\rangle$	$\langle 0.50,pn_2\rangle$	$\langle 1,2,3,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,2,1,1\rangle$	$\langle 0.25,pn_2\rangle$	$\langle 1,1,3,1\rangle$	$\langle 0.0,-\rangle$
$\langle 2,1,2,2\rangle$	$\langle 0.50,pn_3\rangle$	$\langle 1,2,3,1\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,1,1,2\rangle$	$\langle 0.25,pn_3\rangle$	$\langle 1,1,2,2\rangle$	$\langle 0.0,-\rangle$
$\langle 2,1,2,1\rangle$	$\langle 0.50,pn_3\rangle$	$\langle 1,2,2,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,1,1,1\rangle$	$\langle 0.25,pn_3\rangle$	$\langle 1,1,2,1\rangle$	$\langle 0.0,-\rangle$
$\langle 2,2,2,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 1,2,2,1\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,1,3,2\rangle$	$\langle 0.0,-\rangle$	$\langle 1,1,1,2\rangle$	$\langle 0.0,-\rangle$
$\langle 2,2,3,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 1,2,1,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,1,3,1\rangle$	$\langle 0.0,-\rangle$	$\langle 1,1,1,1\rangle$	$\langle 0.0,-\rangle$

Table 2: Ordered constraint combinations with preferences and penalties.

Phase 3 The set of solutions is ranked by preferences and then by penalties (already in Table 2). The solution with highest rank is chosen first. If it turns out not to be feasible, the associated penalty is applied and the next solution is chosen, and so on.

3.3 Mapping SSLA onto SCSP Solvers

Given the our design of an SSLA, mapping it into a SCSP is very easy: variables v_i in the SSLA are mapped onto the corresponding v_i in the SCSP; SSLA domains δ_i are discretized and every discrete identifier is a domain for a SCSP variable; and preferences and penalties (both lattices) are handled together by the def function, so they can be mapped to the A set in a C-semiring with an adequate definition of the def function.

4 Conclusion

We have presented a soft constraint-based framework to seamlessly express QoS properties reflecting both customer preferences and penalties applied to unfitting situations. The application of soft constraints makes it possible to work around overconstrained problems and offer a feasible solution. Our approach makes easier this activity thanks to ranked choices. Introducing the concept of penalty in the Classical SCSP can also be useful during the finding and matching process. We plan to extend this framework to also deal with behavioral penalties.

References

1. C. Miller, A. Ruiz-Cortes, and M. Resinas. An Initial Approach to Explaining SLA Inconsistencies. In *Service-oriented Computing-Icsoc 2008: 6th International Conference, Sydney, Australia, December 1-5, 2008, Proceedings*, page 394. Springer-Verlag New York Inc, 2008.
2. Ugo Montanari. Networks of Constraints: Fundamental Properties and Application to Picture Processing. *Information Sciences 7*, pages 95 – 132, 1974.
3. Rina Dechter. *Constraint Processing*. Morgan Kaufman, 2003.
4. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*. Springer, 2004.
5. Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.
6. S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. *In Proc. IJCAI95*, 1995.
7. Philip Bianco, Grace A.Lewis, and Paulo Merson. Service Level Agreements in Service-Oriented Artchitecture Environment. 2008.

Model-driven Management of Services

Luciano Baresi, Mauro Caporuscio, Carlo Ghezzi, Sam Guinea
Politecnico di Milano
Deep-SE Group - Dipartimento di Elettronica e Informazione
Piazza L. da Vinci, 32 - 20133 Milano, Italy
{*baresi* | *caporuscio* | *ghezzi* | *guinea*}*@elet.polimi.it*

***Abstract*—Applications are increasingly composed of remote services provided by independent parties. This distributed ownership makes it problematic to measure and control quality of service indicators. Management activities must become an integral part of the system's development process, from requirements elicitation, where users identify the quality dimensions of interest, to the implementation, where the actual composition must be paired with suitable means for its run-time management. This paper presents MDMS (Model-Driven Management of Services), a model-driven approach for engineering manageable services. The approach supports the explicit modeling of quality dimensions, management objectives, and key performance indicators, and the transformations required to exploit these concepts at runtime. The methodology is supported by ECoWare, an innovative prototype framework for the deployment and operation of managed services.**

***Keywords*-management; distributed/Internet based software engineering tools and techniques; performance measures; quality analysis and evaluation; model-driven engineering**

I. Introduction

The intertwining of service-oriented infrastructures and flexible business processes is imposing highly dynamic applications. The different parties can change at run time, their ownership is distributed, and they are usually provided through an unreliable communication means (i.e., the Internet). In contrast, the result perceived by the user must be reliable, the application must be robust, and more and more frequently it must also be able to satisfy negotiated service level agreements. All these characteristics can only be offered through a proper and strict run-time management of the application. The execution must be paired with means to probe the applications behavior, analyze it, and react to meet the desired qualities of service and overall objectives. The actual *management* capabilities should be able to work proactively, by analyzing trends and drifts, to anticipate possible problems and change the application before the actual anomalies manifest [1].

Given the strict intricacies of the application, the paper claims that effective management cannot be defined and deployed as a separate entity, but it must be considered a constituent part of the application. The definition of precise management objectives, which are refinable into quantitative KPIs (Key Performance Indicators[1]), must proceed in parallel to the definition of the business logic itself.

The paper proposes a model-driven approach [3], called Model-Driven Management of Services (MDMS), to support this bi-faceted view. The proposal spans from the business perspective down to the actual service-based implementation, always keeping a clear separation of concerns between the functional and the management views. A BPMN (Business Process Modeling Notation [4]) model, along with the first requirements for the actual management, acts as the *Computation Independent Model* (CIM). A SCA (Service Component Architecture [5]) specification, along with a data model, and a model of the key performance indicators of interest, defines the *Platform Independent Model* (PIM). Finally, a *Platform Specific Model* (PSM) that depends on specific service technologies is used to implement the system. MDMS adopts SCA to widen the set of potential services that can be composed, and provide a neutral frame for the actual application.

The paper also presents the tool support provided by MDMS, and in particular ECoWare (Event Correlation Middleware), a general-purpose and extensible infrastructure for managing services. ECoWare is based on two key technologies: Esper [6], which is a Complex Event Processor for correlating and aggregating high loads of events received from different parties, and Siena [7], which is a publish/subscribe middleware for distributing events. Even if MDMS considers the functional capabilities of an application and its management as two sides of the same coin, the separation of concerns fostered by the approach allows for the same functionality to be managed in different ways, for example, according to the needs and requests of different stakeholders. At run time, ECoWare can easily support different management policies concurrently.

Besides presenting the general elements of the approach, the paper exemplifies them on a simple application for loan approvals, rendered as a BPEL process. This choice does not hamper the generality of the proposal, but allows us to reuse Dynamo [8] for collecting run-time data about the process execution.

[1]In this paper, we assume that a KPI is a quantitative measure of the application's capabilities [2]. These indicators are not constrained to the actual performance of the application, but consider different facets. For example, we may have KPIs for an application's reliability, costs, sustainability, security, and data quality.

The rest of this paper is organized as follows. Section II presents an overview of the MDMS approach, while Sections III and IV concentrate on modeling the platform-independent aspects, and on a particular implementation, respectively. Section V surveys some related approaches, and Section VI concludes the paper.

II. Approach Overview

The main objective of the Model-Driven Management of Services (MDMS) approach is to provide a comprehensive MDE solution that considers management throughout an application's lifecycle (i.e., from requirements elicitation up to run time), while always maintaining a clear separation of concerns with respect to its functionality. The application's *functionality* and *management* features are developed in parallel, using different, yet related, models, transformations, and tools. This enables us to specify one functional view for the system, and as many different management views as required, since different stakeholders may identify different QoS dimensions of interest and different management objectives for the same application.

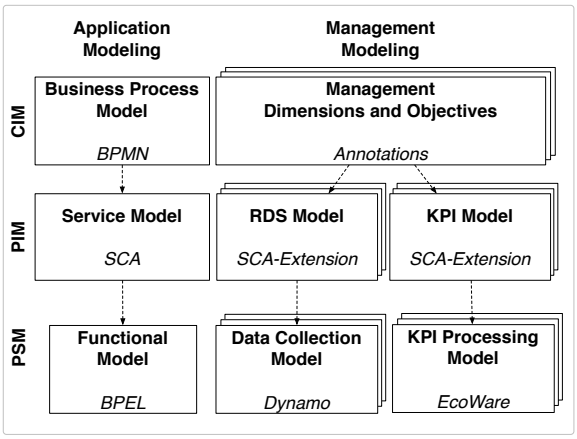


Figure 1. Overview of the MDMS approach.

Figure 1 summarizes the CIM, PIM, and PSM viewpoints of our approach. The first viewpoint consists of the analysis and collection of computation-independent requirements for the managed system. This comprises requirements for its functional design, as well as for the management features it needs to offer. The former are captured using BPMN (Business Process Modeling Notation [4]), while for the latter we propose to use appropriate management annotations. Different stakeholders (providers, clients, etc.) produce, with the help of a requirements modeling expert, different annotations to specify the QoS dimensions they are interested in, and the objectives they have regarding these dimensions[2]. For example, a stakeholder may want a service's response time to be less than 30 seconds, or its reliability to be greater than 95%. Notice that the objectives are assumed to be SMART properties, i.e., Specific, Measurable, Attainable, Relevant and Time-bound [9].

The second viewpoint consists of the platform-independent design of the managed system. MDMS supports the use of the Service Component Architecture's assembly model (SCA [5]) to capture the different parties that need to collaborate to provision the application, what services they will contribute, and the dependencies that exist between them. Regarding management, MDMS maps QoS dimensions to quantitative Key Performance Indicators (KPIs). The Raw Data Sampling (RDS) model defines what data are going to be needed at runtime, while the KPI model defines how these data will be correlated and aggregated to calculate a KPI's value, and how the system will assess if the stakeholder's objective is being reached or not.

The third viewpoint transforms the platform-independent models into platform-specific ones that can be deployed as a running system. Although we base our platform-independent modeling on SCA, a SCA-based execution environment like Apache Tuscany is only an option and not necessarily required. A completely different technological setting is viable as long as it supports the fundamental notions of the service paradigm. MDMS supplies ECoWare to foster management. It is an event-based platform for managing distributed services that can be used in conjunction with different service execution environments. Run-time data, collected through appropriate sensors placed in the execution environment, are run through a network of event processors that incrementally produce the values of the desired KPIs and assess whether the stakeholder's objectives are reached.

A. Running Example

Figure 2 uses BPMN to model the running example we will use in the rest of this paper to demonstrate the MDMS approach. The example describes a simple loan approval business process. The client initiates the process by applying for a loan. If the requested amount is less than 10000$, and the client is determined to be a "low-risk" client, the decision is fast-tracked and approved. If the requested amount is at least 10000$, or the client is not "low-risk", then a thorough assessment is required before providing the final response. Different stakeholders will have different requirements for the application; they will be interested in different QoS dimensions and have

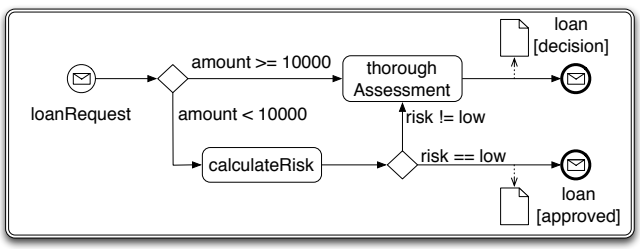


Figure 2. BPMN model of the Loan Approval example.

[2]MDMS does not explicitly focus on SLA management, yet it can be adopted in the context of already existing SLA management frameworks such as the one proposed in the SLA@SOI EU project http://sla-at-soi.eu/

different objectives. On the one hand, a client will be interested in having a service with a reliability of at least 95%. On the other hand, the service provider will surely have a stricter objective and desire a reliability of more than 99%, in order to react promptly before failing to comply to a service level agreement.

III. Platform-Independent Modeling

MDMS offers two distinct, but complementary, meta-models for platform-independent management modeling. The *RDS meta-model* allows designers to define the data sources to be sampled at run time, while the *KPI meta-model* allows designers to specify the Key Performance Indicators to be calculated using the sampled data.

A. The RDS Meta-model

MDMS relies on the assumption that the system to be monitored is modeled by means of SCA. It does not describe the functional properties of the system itself. Rather it represents a complementary and incremental view to the SCA-based functional design.

The cornerstone element of the meta-model is the `ManagedSystem`, which identifies the system under development. It specifies the `ServiceComponents` that constitute the system, and that represent the main `ManagedElements` of interest. Each ServiceComponent exposes a set of `ServiceComponentActions` that implement its `UnitOfWorks` –i.e., its functional elements. A functional element can be classified either as an `InternalAction` or an `OperationCall`.

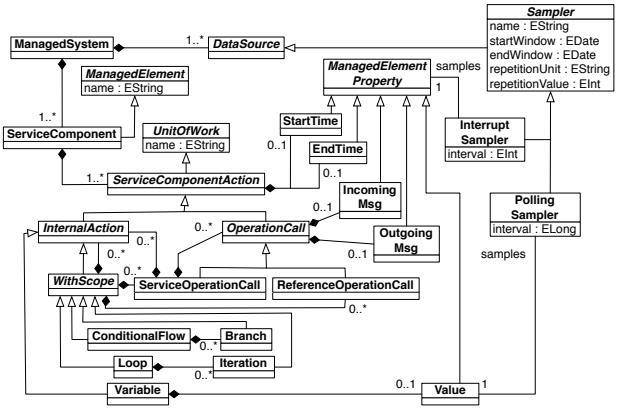


Figure 3. The RDS meta-model

An internal action identifies the elements that internally implement the component's behavior. In our models we distinguish between internal actions that define a scope (`WithScope`), and those that do not. An internal action defines a scope if its semantics conceptually introduce a "code" block in which other actions can be placed – e.g., the `Loop` action can contain other internal actions or operation calls. This distinction allows designers to express the need to monitor and analyze a specific action in the context of a specific scope. Among the possible internal actions, Figure 3 shows a few key examples, namely conditional flows (`ConditionalFlow`), loops (`Loop`), and variables (`Variable`). However, the RDS meta-model is extensible and further actions can easily be added.

`OperationCalls` are used to specify the elements that concern the component's interactions with external partner components. Due to the bilateral nature of these interactions, the RDS meta-model distinguishes between provided (`ServiceOperationCall`) and required (`ReferenceOperationCall`) operation calls.

The ServiceComponentActions will contribute raw data to the calculation of the KPIs of interest. The data are identified by `ManagedElementProperties`, which can be general in nature (`StartTime` and `EndTime`) or ServiceComponentAction-specific. For example, operation calls can identify incoming (`IncomingMsg`) and outgoing `OutgoingMsg` messages, as well as start and end times, while variables can identify their value (`Value`).

`Samplers` define how to collect the raw data and provide them for further processing. To this extent, the RDS meta-model currently provides two kinds of samplers. `InterruptSamplers` monitor the property of interest and provide it for processing if and only if the its status changes. These samplers support the notion of interval to limit their outputs –e.g., output once every x changes in the property's status. `PollingSamplers` check a property and provide it for processing periodically. They also support a notion of interval to limit their outputs –e.g., output once every x minutes. Both kinds of samplers can be further configured to be active only within certain time windows. A time window is defined by a start date (`startWindow`) and an end date (`endWindow`), and can be repeated using the `repetitionUnit` and `repetitionValue` attributes. For example, a time window can start on Monday at 8AM and end on Monday at 10AM, and be repeated every week (`repetitionUnit = week, repetitionValue = 1, startWindow = Monday 08.00AM, endWindow = Monday 10.00AM`). If no activation window is specified, the default is to consider the sampler as always active.

B. The KPI Meta-model

While the RDS meta-model allows designers to specify the sampling of raw data, the *KPI meta-model* specifies how the raw data can be correlated and aggregated to produce the desired performance indicators, and how these indicators can be analyzed.

Referring to Figure 4, all the data that are processable in the model are obtained through a `DataSource`. Samplers are one example of a DataSource; `DataProcessors` are another. They read data from one or more sources, process them and, if needed, provide the processed data as a new source. This mechanism allows designers to combine different DataProcessors in a pipe-and-filter fashion to achieve more complex data processing.

The KPI meta-model defines two different kinds of DataProcessors: `KPIs` and `Filters`. KPIs read data from one (or more) sources, combine them by means of well-

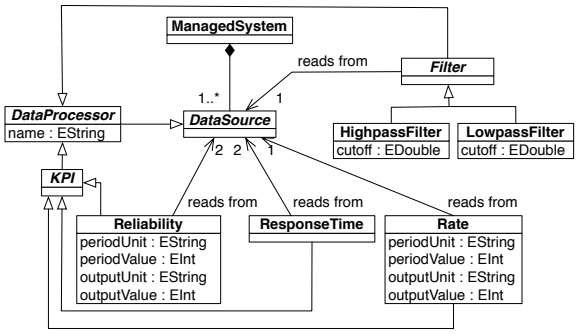


Figure 4. The KPI meta-model.

defined rules, and provide their output as a new data source. In the model we introduce the `ResponseTime` KPI, which computes the time elapsed between two time instants (e.g., the end time of a service operation call and its corresponding start time), and the `Reliability` KPI, which computes the number of correct interactions with a service over the total number of interactions attempted. When using a Reliability KPI the designer must also state (i) over how much time it has to be calculated, and (ii) how often its output value needs to be made available. For example, the designer might want to calculate a service's reliability considering the last 12 hours, and output a new value every 5 minutes. This is achieved through its `periodUnit` and `periodValue`, and `outputUnit` and `outputValue` attributes respectively.

Filters read data from a single source and provide them as a new source if they satisfy a well-defined constraint. These are useful to define simple KPI analyses. For example, if we want to be notified when a response time is higher than a given threshold we can use a `Highpass Filter`, and if we want to be notified when a service's reliability drops below a given threshold we can use a `Lowpass Filter`.

New data processors can easily be added to the KPI meta-model through extension mechanisms. For each new data processor we must also provide an ECoWare specific processing component, as we shall see in § IV-B.

C. Modeling the Loan Approval Process for Management

Starting from the BPMN model presented in Section II-A, we need to determine how many parties/services need to be involved to effectively provide the overall composite service. Figure 5 illustrates the SCA-based functional design of our running example. Service operation calls are shown as white arrow blocks, while reference operation calls are shown as grey ones. In this example `loanApproval` is responsible for providing the service's `loanRequest` service operation call. It has two dependencies that are satisfied by the `assessment` and `riskCalculation` services.

Once the platform-independent application model is complete, we can model the raw data sampling and the KPIs. Figure 6 shows how to model the reliability of loanApproval, and how to use a filter to find out if its

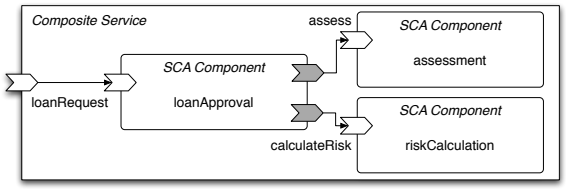


Figure 5. The platform-independent application model.

value drops below a given threshold.

The RDS model is presented on the figure's left-hand side. To calculate the reliability the system needs to know the number of requests that are issued to the loanApproval, and how many of them result in failures. To this end it is sufficient to sample the service component's start and end times. In particular we want to sample the start and end times of the approve service operation call provided by the loanApproval ServiceComponent, every time it is executed. This is why we use two interrupt samplers, both setup with an interval value of 1. Since we want to leave the samplers always on, we do not specify any activation window.

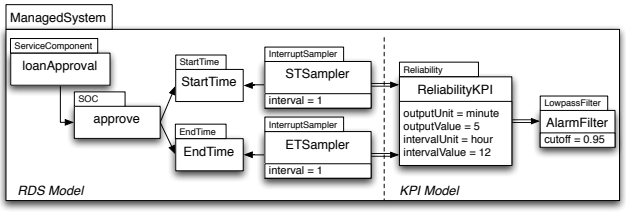


Figure 6. The Reliability model.

The KPI model is presented on the right-hand side. In it we use a Reliability KPI that reads data from the `STSampler` and the `ETSampler`. It is configured to calculate the KPI considering the last 12 hours (intervalUnit = "hour" and intervalValue = "12"), and to output a new value every 5 minutes (outputUnit = "minute" and outputValue = "5"). Finally we use a lowpass filter that reads the produced reliability values and ignores those that are higher than the 0.95 threshold.

IV. Platform-Specific Implementation

In this section we present ECoWare (Event Correlation Middleware), an event-based platform for managing distributed services. It consists of Esper-based event processing components that collaborate through Siena, a publish/subscribe event notification service (see Figure 7). Esper provides an Event Processing Language (EPL) that allows developers to easily define complex event conditions, correlations, and aggregations, thus effectively minimizing the effort required to keep track of a distributed system's behavior. Siena is a publish/subscribe event notification service implemented as a distributed overlay-network suitable for large numbers of communicants and high volumes of events. We use Esper to implement the DataProcessors of our KPI models (see

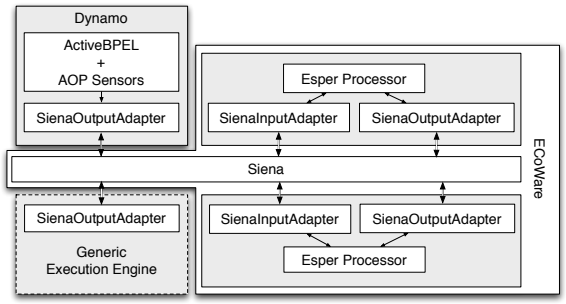


Figure 7. The ECoWare Architecture.

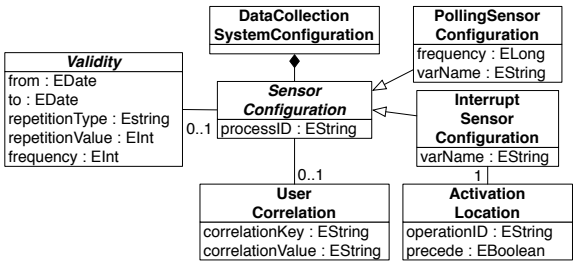


Figure 8. Dynamo's data collection configuration meta-model.

§ III-B), while the loose-coupling provided by Siena ensures that processors can be dynamically combined in a pipe-and-filter fashion. ECoWare defines a normalized event format for the data that flow through Siena, and provides configurable adapters (`SienaOutputAdapter` and `SienaInputAdapter`) that allow the system to be used in conjunction with different external service execution environment.

In our ongoing example, we transform the platform-independent service model into a BPEL implementation of the loanApproval service, with assessment and riskCalculation being offered as external services. This decision allows us to use Dynamo [8] as our execution environment, and to leverage its data collection capabilities. Dynamo is a feature-enriched execution environment for BPEL processes. It leverages AOP technology to extend ActiveBPEL and instrument processes with data collection, data analysis, and recovery activities. The use of AOP ensures a clear separation of concerns between business and management logic, and allows different stakeholders to define different management activities for the same process. We have equipped Dynamo with a `SienaOutputAdapter` to translate the internal format it uses for the data it collects into a normalized ECoWare event, and to publish it to Siena.

A. Data Collection

Dynamo extends ActiveBPEL with data collection capabilities that are implemented through two kinds of sensors: interrupt and polling sensors. ActiveBPEL exploits the visitor pattern on an in-memory tree representation of the executing process, one in which each node implements a specific BPEL activity (e.g., an invoke, a receive, an assign, etc.). Interrupt sensors intercept the process' execution before or after the visitor's method is called on a specific node in the tree, while polling sensors intercept the process' instantiation and attach a polling thread to a specific BPEL variable.

The meta-model in Figure 8 describes the information needed to configure these sensors. All sensor configurations, regardless of their type, contain a `processID`, a `Validity`, and a `UserCorrelation`. The processID is a unique identifier of the process deployed within Dynamo. The Validity defines a time-frame within which the sensor is to be considered active. `from` and `to` are two dates that define the time-frame, while `repetitionType` and `repetitionValue` define the frequency with which the time-frame is repeated. A validity also uses a `frequency` to state how many process instances, within the temporal window of activation, need to be executed between two successive data collections. If no validity is given, Dynamo's default behaviour is to keep data collection active at all times. The UserCorrelation supports the deployment of different data collection specifications for different process clients. The `correlationKey` is the name of an XML element that is present in the SOAP message that initiates the process (e.g., "clientName"), and the `correlationValue` is the value we expect to find therein at run time (e.g., "name"). These data are checked every time a new process is instantiated to determine whether the sensor needs to be instantiated. If no UserCorrelation is specified Dynamo's default behaviour is to activate the sensor on all instances of the process.

The configuration of an interrupt sensor adds an optional `varName` and an `ActivationLocation`. The varName parameter identifies a specific BPEL variable that has to be collected. Note that sensors always provide a timestamp[3], even if no varName is given. The ActivationLocation indicates the point in the process in which the sensor needs to be activated. It contains an `operationID`, a unique identifier of a BPEL activity within the process, and a `precede` parameter that states whether the sensor needs to be activated prior to, or after, the operation is execution. The configuration of a polling sensor adds a mandatory `varName`, and a `frequency` (expressed in seconds) with which to perform the polling of the variable's value.

We transform RDS models into Dynamo data collection configuration models using QVT (Query/View/-Transformation) [11], a standard for model transformation proposed by the Object Management Group (OMG). In particular, we use the Relations meta-model to declaratively define the relationships that must hold between the

[3]We use NTP (Network Time Protocol) timestamps generated using Apache's Commons Net [10] implementation. It guarantees a synchronized clock when sampling sources on different computers. Its precision is in the order of 10 milliseconds, which is acceptable given the typical services deployed in ECoWare.

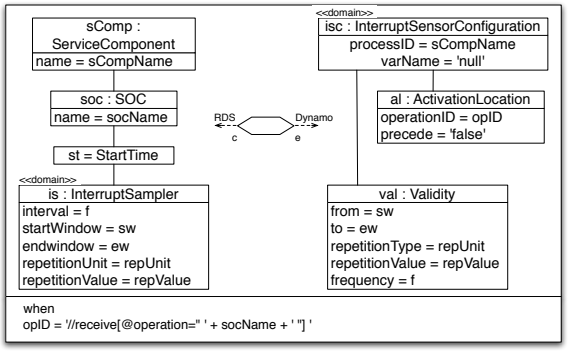


Figure 9. Example of a QVT-based transformation between models.

source RDS model and the target Dynamo model. Due to the complexity and length of the transformation code it is impossible for us to report it here in its entirety. Instead we will concentrate on the transformation of a simplified example, one in which a StartTime Sampler is applied to a service operation call, and use QVT Relations' graphical syntax to give the reader a flavor for our transformations. Figure 9 enforces an interrupt sensor configuration in which the process (`sCompName`) is intercepted after (`precede = 'false'`) its service operation call (`socName`)'s corresponding BPEL receive activity (`opID`) is executed. Moreover, since we are dealing with a StartTime, the configuration does not require a varName. Note that MDMS does not explicitly acknowledge multi-tenancy. If multi-tenancy is needed, the UserCorrelation values need to be added to Dynamo's configuration by hand.

Dynamo's data does not directly conform to ECoWare's normalized event format, which requires that all events contain an `eventType`, an `originID`, and an `instanceID`. We resolve this by attaching Dynamo-specific SienaOutputAdapters to our sensors. The eventType is used to indicate the kind of event being sent. For example, in the case of an interrupt sensor it is calculated by looking at the sensor's operationID, precede value, and varName. Sensors on receive activities produce StartTime or IncomingMsg event types, depending on the presence of a varName. The same is true for sensors on reply activities, which produce EndTime or OutgoingMsg event types. Sensors on invoke activities produce StartTime, EndTime, IncomingMsg or OutgoingMsg event types, depending on their varName and precede values. In the case of a polling sensor the eventType is always a Value. The originID identifies the sensor producing the event, and is used to ensure event routing in Siena. The instanceID identifies the service instance that produced the contents of the event. It is generated concatenating the name of the process being executed, the name of the BPEL activity to which the sensor is attached, and the process instance id generated at run time by ActiveBPEL. The resulting value is used by Esper processors to correlate events that come from different sources. Special care is given to asynchronous messaging, since in these cases we have to correlate receive-reply and invoke-receive pairs that have different activity names; in the first case we use the name of the receive activity, in the second case we use the name of the invoke activity.

B. KPI Processing

Complementary to the platform-independent RDS model, the KPI model specifies "what" KPI needs to be produced and analysed, and "who" is involved in producing and analysing it. In our example we want to analyze the reliability of loanRequest, calculated over the last twelve hours, and be notified when it drops below 95% (i.e., the "what"). This is achieved composing a basic `Reliability` KPI and a `LowpassFilter` (i.e., the "who"s).

QVT is used to define and perform the transformations between the KPI model and the Esper configuration model. We do not discuss the transformations here, since MDMS and ECoWare were conceived together and are conceptually aligned. Suffice to say, in our example the Reliability KPI is configured using the periodUnit, periodValue, outputUnit, and outputValue attributes present in the platform-independent KPI model (see § II-A), while the Lowpass Filter is configured using the cutoff attribute. Instead we will concentrate on how the KPI data processors are concretely implemented using Esper.

Esper provides an Event Processing Language (EPL) for defining complex event conditions, correlations, and aggregations. EPL is an SQL-like language with `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` and `ORDER BY` clauses. The difference between SQL and EPL is that instead of running queries against stored data, Esper stores queries and runs data through them. The execution model is thus continuous rather than limited to the exact moment at which the query is submitted.

EPL provides key constructs for building data views over events, and for analyzing event arrivals with respect to time constraints. Specifically, an EPL statement is defined exploiting the following concepts: (i) *windows* are used to specify how many past events need to be considered by Esper when processing EPL statements and, (ii) *streams* are used to inform appropriate listeners about events that enter (*insert stream*) or leave a window (*remove stream*).

As discussed in Section IV-A, Dynamo samples the required data and sends them as Siena events. A SienaInputAdapter is used by the ReliabilityKPI to subscribe to the events whose originID is either STSampler or ETSampler, and to transform them into `StartTime={originID, instanceID, timestamp}` and `EndTime={originID, instanceID, timestamp}` respectively.

Reliability is generically defined as $R = 1 - \frac{Total_f}{Total_r}$, where $Total_f$ is the amount of failures associated with the loanRequest operation in a given amount of time, and $Total_r$ is the total number of requests made to the loanRequest operation in that same time frame. The KPI is therefore implemented using two EPL statements, one for $Total_f$, and one for $Total_r$. Both use a 15 seconds time-

window, i.e., a sliding window that extends 15 seconds into the past to limit the events they need to keep track of. The 15 seconds represent the amount of time we are willing to wait for a response from the service before we decide that it failed[4]. The first EPL statement is used to calculate the total number of requests:

```
(1) SELECT * FROM StartTime.win:time(15sec)
```

An Esper listener receives new data as soon as the engine processes events for such statement. Its reaction is to accumulate the number of events that occurred within the window, and update the $Total_r$ value calculated over the last 12 hours (i.e., `historyUnit + historyValue`).

The second EPL statement is used to calculate the total number of failures that occur:

```
(1) SELECT rstream * FROM StartTime.win:time(15 sec) ST
(2)   FULL OUTER JOIN EndTime.win:time(15 sec) ET
(3)         ON ST.instanceID = ET.instanceID
(4)   WHERE ET.callID is null
```

Specifically, we define a failure as the incapability to match two corresponding StartTime and EndTime events within a 15 second time window. The EPL statement uses the "rstream" keyword to select events from the remove stream. This means it would tend to notify its listener every time an event exits the time window. However, it should only notify its listener if a StartTime is exiting and no corresponding EndTime can be found. The fact that an EndTime is missing is discovered thanks to the outer join (line 2) performed on the instanceID attribute of both events. In Esper, if an outer join cannot make a match (line 3), a result is presented nevertheless, and the default value `null` is given in place of the missing attributes (line 4). The listener, once again, simply counts the number of notifications it receives, and updates the $Total_f$ value calculated over the last 12 hours.

To conclude, the two listeners collaborate once every 5 minutes to calculate the new reliability (i.e., `outputUnit + outputValue`), and update their $Total_r$ and $Total_f$ values to ensure the next reliability value will not consider requests or failures that have occurred more than 12 hours ago. As soon as a new reliability value is available it is disseminated as a Siena event whose eventType is `Reliability`; this event contains an internal attribute called `value`.

In our example, the Reliability events are run through a LowpassFilter, and therefore through a further ESP statement. In this case the component compares the reliability event's internal value with the value of the cutoff taken from the model.

```
(1) SELECT * FROM Reliability(value<0.95).win:length(1)
```

The statement uses a length window of size 1, meaning that only 1 event will be stored in the window at a time. Only events whose internal value is less than 0.95 are allowed to enter the window; every time this occurs the old event is discarded and the component's listener is notified of the new arrival.

[4]15 seconds is only an indicative value; in real case studies, it should be equal to the "timeout" value set up by the service provider.

C. Early evaluation

To run our experiments we set up a Siena overlay network composed of four server nodes and two clients, namely a publisher (i.e., the Dynamo sampler) and a subscriber (i.e., the Esper engine). The Dynamo sampler is in charge of collecting data and publishing them as events, while the Esper engine receives and processes them to calculate the KPIs. The tests were run on a MacBook with a $2.4Ghz$ Intel Core 2 Duo processor and $4GB$ of RAM running Mac OS X version 10.6.4.

Due to the simplicity of such a case study, in terms of both the number of events that are delivered and processed, the overhead introduced by Siena and Esper is negligible with respect the overhead introduced by Dynamo for collecting data. In fact, both Siena and Esper have been designed and implemented to address scalability – i.e., to manage large amount of events, in terms of forwarding [12] and processing [13], respectively.

Table I
PERFORMANCE IMPACT IN MILLISECONDS DUE TO DYNAMO.

Setup	Collection time		
	avg	min	max
ActiveBPEL	21.02	15	216
Dynamo_0_Sensors	32.66	23	308
Dynamo_1_Sensor	34.02	25	364
Dynamo_2_Sensors	35.75	26	386

Table I summarizes the impact that the installation of sensors has on the execution time of our simple example. We setup four different tests and ran each 1000 times. The first test was performed on a standard ActiveBPEL engine, the second used Dynamo with no sensors installed, the third used Dynamo with one sensor, and the last one used Dynamo with two sensors. As we can see Dynamo introduces an average overhead of 11 milliseconds, regardless of the number of sensors installed, and each sensor introduces an extra average overhead of 2 milliseconds. This is perfectly aligned with the more detailed analyses we have performed in [14].

V. RELATED WORK

Management has been a hot topic for standardization by part of different consortiums. Oasis has proposed the Web Services Distributed Management (WSDM) [15], a protocol for the interoperability of management information and features. It focuses on two main aspects: how to use web service technology as the foundation of a resource management framework, and how these notions can be adapted to Web services themselves. An alternative standard, called WS-Management [16], has been proposed by the Distributed Management Task Force (DMTF). Its goal is similar to WSDM's, as it provides support for generic resource management using Web service standards. It also proposes a special binding for managing resources that are defined using their Common Information Model (CIM).

Different MDE methodologies for managing extra-functional properties of service compositions have been proposed: Chowdhary et al. [17] address the specification

of business-level performance indicators and their direct transformation to platform specific models, Debusmann et al. [18] define SLA parameters together with the specific indicators needed within the SLA itself, and Chan et al. [19] address the automatic generation of component-based instrumentations for monitoring specified QoS concerns. The work that most resembles our own is that of Christof et al. [20]. The main difference with our work lies in the different level of abstraction proposed. Our meta-models provide an extensible set of high-level off-the-shelf KPIs. The definition of the KPIs themselves is an issue for platform-specific implementations. In their work, the definition of the KPI is something that needs to be modeled; to this end they propose lower-level processing operations (e.g., addition, subtraction, etc.) for defining the calculation that needs to be performed. We believe that this level of detail in the modeling makes matters unnecessarily complex, especially with KPIs that are stochastic or more fuzzy in nature. Another key difference lies in the nature of the proposed management infrastructure. Instead of being centralized, ECoWare's solution is event-based and composed of loosely-coupled elements.

VI. Conclusions and Future Work

This paper claimed that the effective management of service-oriented applications cannot be defined and deployed as a separate entity. Rather, management must be a constituent part of the application itself, and addressed starting from the early stages of the development process. MDMS aims to provide a comprehensive model-driven approach that considers quality dimensions and management objectives as first-class elements that must be properly addressed. The paper also presented ECoWare, the distributed management framework developed to support MDMS, and exemplified the overall approach through a simple loan approval application. The first experiments were encouraging and demonstrated the feasibility of the approach. We will continue to refine the approach and develop the tools necessary to fully automate it. We will also further investigate the actual management of heterogeneous service-oriented applications, and the integration of advanced management capabilities with ECoWare.

Acknowledgment

This research has been funded by the European Commission, Programme IDEAS-ERC, Project 227077-SMScom (http://www.erc-smscom.org), FP7 Integrated Project 216556-SLA@SOI (http://sla-at-soi.eu/), and the Network of Excellence S-Cube (http://www.s-cube-network.eu/)

References

[1] M. P. Papazoglou and W.-J. v. d. Heuvel, "Web services management: A survey," *IEEE Internet Computing*, vol. 9, no. 6, pp. 58–64, 2005.

[2] D. Parmenter, *Key Performance Indicators – Developing, implementing, and using winning KPIs*. John Wiley Sons, 2007.

[3] D. Schmidt, "Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, Feb 2006.

[4] Object Management Group, "Business Process Model And Notation (BPMN) 1.2," http://www.omg.org/spec/BPMN/1.2/, 2009.

[5] OpenSOA, "Service component architecture specifications," http://www.osoa.org, 2007.

[6] EsperTech, "Complex event processing," http://esper.codehaus.org, 2010.

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, pp. 332–383, Aug. 2001.

[8] L. Baresi and S. Guinea, "Self-supervising bpel processes," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2010.

[9] G. T. Doran, "There's a S.M.A.R.T. way to write managements's goals and objectives," *Management Review*, vol. 70, no. 11, 1981.

[10] Apache, "Network Time Protocol Provider 1.0," http://directory.apache.org/apacheds/1.0/ntp-protocol-provider.html.

[11] Object Management Group, "Query/View/Transformation Version 1.1 - Beta 2," http://www.omg.org/spec/QVT/1.1/Beta2/, 2009.

[12] A. Carzaniga and A. L. Wolf, "Forwarding in a content-based network," in *Proceedings of ACM SIGCOMM 2003*, Karlsruhe, Germany, Aug. 2003, pp. 163–174.

[13] EsperTech, "Esper performance," http://docs.codehaus.org/display/ESPER/Esper+performance, 2007.

[14] L. Baresi and S. Guinea, "Self-supervising BPEL Processes," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2010.

[15] OASIS, "Web Services Distributed Management (WSDM)," http://www.oasis-open.org/specs/, 2006.

[16] Distributed Management Task Force, "Web Services for Management," http://www.dmtf.org/standards/wsman/, 2010.

[17] P. Chowdhary, K. Bhaskaran, N. S. Caswell, H. Chang, T. Chao, S.-K. Chen, M. Dikun, H. Lei, J.-J. Jeng, S. Kapoor, C. A. Lang, G. Mihaila, I. Stanoi, and L. Zeng, "Model driven development for business performance management," *IBM Syst. J.*, vol. 45, no. 3, pp. 587–605, 2006.

[18] M. Debusmann, R. Kröger, and K. Geihs, "Unifying service level management using an mda-based approach," in *NOMS*, 2004, pp. 801–814.

[19] K. Chan and I. Poernomo, "Qos-aware model driven architecture through the uml and cim," *Enterprise Distributed Object Computing Conference, IEEE International*, vol. 0, pp. 345–354, 2006.

[20] C. Momm, M. Gebhart, and S. Abeck, "A model-driven approach for monitoring business performance in web service compositions," in *ICIW '09: Proceedings of the 2009 Fourth International Conference on Internet and Web Applications and Services*, Washington, DC, USA, 2009, pp. 343–350.

Proactive SLA Negotiation for Service Based Systems

Khaled Mahbub and George Spanoudakis
School of Informatics
City University London, UK
{K.Mahbub, G.Spanoudakis}@soi.city.ac.uk

Abstract

In this paper we propose a framework for proactive SLA negotiation that integrates this process with dynamic service discovery and, hence, can provide integrated runtime support for both these key activities which are necessary in order to achieve the runtime operation of service based systems with minimised interruptions. More specifically, our framework discovers candidate constituent services for a composite service, establishes an agreed but not enforced SLA and a period during which this pre-agreement can be activated should this become necessary

1. Introduction

A service level agreement (SLA) is an explicit contract between the provider and the consumers of a service that defines the quality and, sometimes, functional properties which should be guaranteed during the provision of the service, as well as the penalties that should be applied in case of defaulting [7][10][11]. An SLA is set through a negotiation process between the provider and the consumer of a service [4][12]. This process is particularly complex in the case of composite services since, in order to ensure that the provision of a composite service S is in line with the SLAs required by its clients, the provider of S should also negotiate and establish subordinate SLAs with the providers of the constituent services of S. Furthermore, when a constituent service of S becomes unavailable at runtime or fails to perform according to its SLA, the provider of S should be able to discover alternative replacement services for it and negotiate SLAs with them at runtime.

As it has been suggested in [15], to minimise the runtime interruption in the provision of composite services, the discovery of back up replacement services for their constituents should be proactive, i.e., it should be performed before a constituent service of S becomes unavailable or fails to perform according to its established SLA. Proactiveness is important since service discovery is a time consuming activity and, therefore, carrying it in a reactive mode, is likely to cause significant interruption in the provision of the composite service and violations of its own SLAs. SLA negotiation should also be proactive as it will be necessary to have adequate SLAs for the potential replacement services that have been identified by proactive discovery attempting SLA negotiation just prior to binding to an alternative service is likely to cause significant delay.

Existing work on service level agreements has focused on SLA specification [13][14], negotiation [4][6] and monitoring [8]. The need for runtime SLA negotiation or re-negotiation has been realised in [2][3][5][9], where either the terms of an SLA are revised to accept a constituent service from an existing provider [2][5] or a new SLA is negotiated with a new service provider and an existing SLA is terminated [3]. All these approaches, however, are reactive as they support corrective actions only after an SLA has been violated. Thus they can fail to guarantee uninterrupted runtime provision of composite services.

To address this shortcoming, in this paper we introduce an approach for proactive runtime SLA negotiation. Our approach is based on an extension of a tool for proactive runtime service discovery described in [15] enabling it to support proactive SLA negotiation as part of the discovery process. More specifically, our approach weaves SLA negotiation into runtime service discovery and provides a clear process model for carrying these two activities in a coordinated manner. It also leverages upon the language for expressing runtime service discovery queries that has been developed in [15] and extends it in order to enable the specification of SLA negotiation criteria. Thus, our approach provides integrated runtime support for both proactive service discovery and SLA negotiation that is necessary for achieving composite service provision with minimised interruptions at runtime.

Proactive SLA negotiation is weaved into the discovery process and is performed after the execution of service discovery queries to ensure that adequate SLAs can be set for the discovered services. The objective of proactive negotiation is to establish an agreed but not enforced SLA and a period during which the consumer of the service will be able to activate the pre-agreement should this become necessary. Following this, a discovered service can be considered as a candidate constituent service for a composite service. The negotiation process is also repeated when a pre-agreed SLA comes close to expiry and, therefore, would cease to be binding for the provider unless renegotiated.

The rest of this paper is structured as follows. In Section 2, we discuss the architecture of the framework for integrated proactive runtime service discovery and SLA negotiation. In Section 3, we describe the negotiation process. In Section 4, we provide an overview of the language for specifying the rules for triggering and carrying out the SLA negotiation process (SLA triggering and SLA negotiation rules). In Section 5, we review related work and finally in Section 6, we provide some concluding remarks and outline directions for future work.

2. Overview of proactive service discovery and SLA negotiation framework

The architecture of the integrated service discovery and SLA negotiation framework is shown in Figure 1. According to the figure, the framework consists of a runtime service discovery tool, a service listener, an SLA negotiation broker and a monitor. It also interacts with external service registries and event captors.

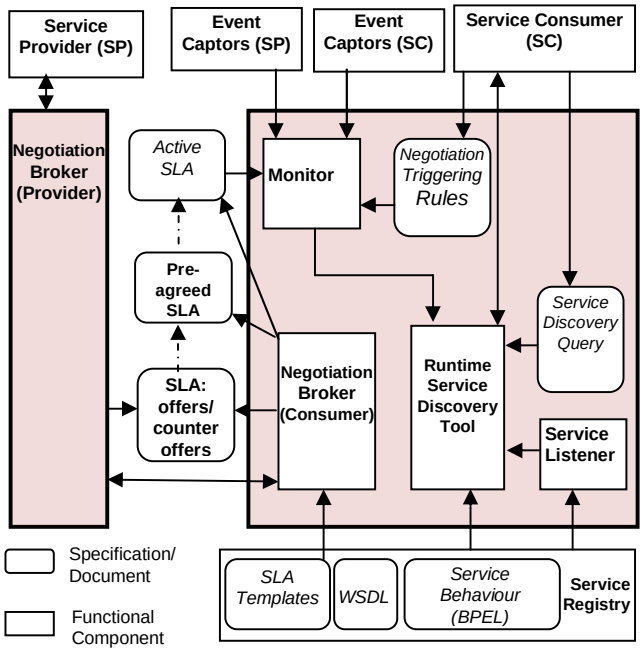


Figure 1. Architecture for proactive (and reactive) SLA negotiation

The *runtime service discovery tool* is used to identify potential alternative services for the services that a composite service uses currently. The discovery process is driven by *service discovery queries*. These queries are associated with each of the constituent services S_c of the composite service S and specify the conditions that should be satisfied by any service that could replace them in the composition. These conditions can refer to the structural (interface), behavioural, contextual, and quality characteristics that services should have in order to be acceptable replacements for S_c and, therefore, provide the criteria for discovering candidate constituent services for S_c. Service discovery queries can be executed in two modes: (a) in a reactive mode where the query is executed when the constituent service S_c it is associated with becomes unavailable or fails to satisfy an agreed SLA and, therefore, a replacement service should be identified for it, or (b) in a proactive mode where the query is executed in parallel with the operation of the composite service S in order to discover and maintain a set of candidate replacement services for its constituent services. In the proactive execution mode, the query is executed initially to build a replacement set of services for S (*RS*) and then anytime when an event indicating that the description of some service in *RS* has been changed or a new service that could be a candidate for inclusion in *RS* has emerged.

The *negotiation broker* is the component that manages and executes the negotiation process on behalf of a service consumer (i.e., the composite service) or a service provider. Our architecture assumes that a separate instance of the negotiation broker is associated with each of the two sides (the service provider and consumer) that participate in the negotiation process. Negotiation brokers are responsible for negotiating and agreeing the guarantee terms of an SLA. The negotiation process can be either reactive or proactive. In proactive negotiation, the negotiation process is carried out according to a two-phase protocol that may result in a provisionally agreed but not activated SLA (see *Pre-agreed SLA* in Figure 1) or negotiation failure. In reactive negotiation, the negotiation process is executed according to a single phase protocol that can result in an agreed and activated SLA (see *Active SLA* in Figure 1) or negotiation failure. In the framework, a *pre-agreed SLA* describes a service level agreement that has been reached but not activated yet. Pre-agreed SLAs have an expiry period within which they will have to be activated or cease to exist. A pre-agreed SLA becomes an *Active SLA*, if the consumer of the service decides to activate it.

The *service registry* contains descriptions of services. These should include at least a specification of the interface of the service (WSDL) and *SLA templates* indicating the terms (e.g. service quality levels, costs etc) under which the provider of service is typically willing to provide it. Additional types of service descriptions that are supported by the framework are models of service behavior expressed in BPEL and further quality characteristics that might not be included in the existing SLA templates of the service or complement them by specifying the possible range of values for a given characteristic as opposed to the individual quality level points or sub-ranges that are specified in the SLA templates of a service.

The *service listener* polls service registries regularly to identify changes in existing service descriptions or new services that might have become available.

The *monitor* is responsible for checking at runtime whether the provision of a service by a given provider and the use of it by given service consumer are in line with established SLAs. In general there are two monitors: one associated with the service provider and another associated with the service consumer[1]. A monitor at either of these two sides is typically used to detect if the SLA guarantee terms which should apply to the provision of the service are satisfied, and whether the conditions of the negotiation triggering rules of the relevant party are satisfied in order to generate signals for triggering negotiation (whether proactive or reactive). A monitor at the side of the provider may also check the levels of service usage by the relevant consumer as the latter may be preconditions of SLA guarantee terms (e.g., a service provider may have agreed to an average service throughput only if the rate of service calls by a particular provider does not exceed a given threshold).

If a monitor detects (or forecasts) that the conditions of negotiation triggering rules in the negotiation policy of a service provider or consumer are (or will be) violated, it will inform the relevant negotiation broker to initiate a negotiation or renegotiation.

The checks performed by the monitors take into account events that are intercepted during the use of services (e.g., service invocations and responses, server loads). These events are intercepted and notified to the framework by different types of *event captors* that may be associated with different services (e.g. SOAP message captors). These events are notified to the monitor for verifying the adherence of services to different SLA guarantee terms and checking whether some SLA negotiation activity should be initiated.

Negotiation Triggering Rules determine the circumstances under which the negotiation of new service level agreements should start (e.g., when a provisionally agreed SLA is about to expire). Separate sets of such rules may be specified by service providers and consumers for this purpose. The negotiation triggering rules are monitored once an SLA is established.

3. SLA negotiation process

Figure 2 presents the service discovery process of the framework with the activity of SLA negotiation embedded within it. The integrated process is specified as a UML activity diagram.

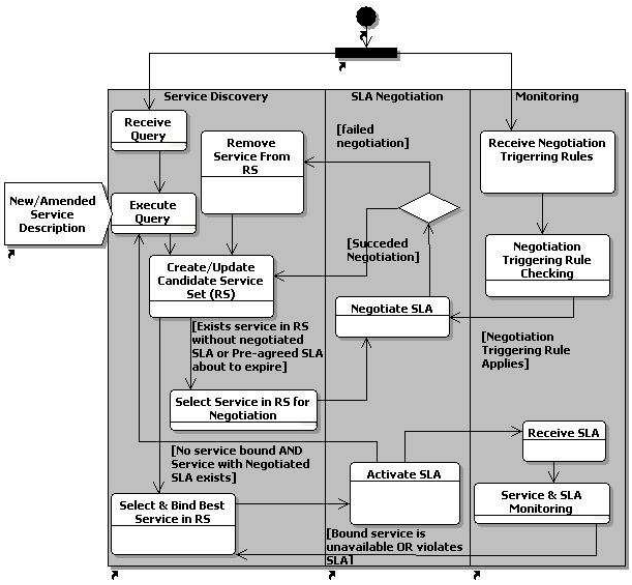


Figure 2. Integrated service discovery and SLA negotiation process

The process starts with the submission of a service discovery query by the composite service (i.e., the consumer of constituent services). As discussed in Section 2, a query can specify different service discovery criteria, namely: (a) structural criteria describing the interface of required services, (b) behavioural criteria describing the functionality of required services, and (c) constraints describing quality characteristics of required service. The initial execution of the service discovery query (see the action state *Execute Query* in Figure 2) results in a list of potential candidate services (*RS*). The candidate services are identified by evaluating the structural, behavioural and quality characteristics specified in a query against the structural, behavioural and quality of service specifications in service registries. The execution of the discovery query also computes distances between a query and candidate services based on the query criteria, and ranks the candidate services based on their distances to the query. The list of potential candidate services is updated by executing the service discovery query when the framework is informed by the service listener that a new service has become available in a registry or the description of an existing service has been modified (see the signal accepting state *New/Amended Service Description* in Figure 2). This ensures that new or updated services are considered by the process.

Once an initial set of candidate services has been built or updated (see the action state *Create/Update Candidate Service Set*), the framework selects a service that does not

[1] It is, however, also possible that the monitors of two parties of an SLA are realised by a same monitoring service which may be offered by a trusted external third party. Such a shared monitoring service would in general be monitoring different sets of rules for each of the involved parties and based on different sets of events.

have a negotiated SLA from *RS* for negotiation (see the transition guarded by the condition *Exists Service in RS without Negotiated SLA*).

Subsequently, in the negotiation phase (i.e., *Negotiate SLA*), the desired level of service is negotiated with the selected candidate service. In this phase, the *QoS* characteristics of each candidate service are negotiated in order to achieve the best possible SLA that is within the boundary constraints of the two parties for each of these services. Negotiation during this phase may fail. If this happens for a selected candidate service, the service is removed from *RS* and a new negotiation will start with another candidate service in *RS* that does not have a pre-agreed SLA. If the negotiation with a selected service succeeds, however, a provisional SLA is established and the selected candidate service in *RS* is updated to flag the existence of the pre-agreed SLA.

It should be noted that the negotiated SLAs for the services in *RS* do not come into force immediately. For each pre-agreed SLA, the negotiation process establishes a time period over which the pre-agreed SLA can be automatically brought into force without further negotiation. This will happen if the relevant service is selected for binding to the composite service. If the validity period of a pre-agreed SLA comes close to expiry without the candidate service being bound to the composite service, the framework will proactively re-negotiate the SLA – see the transition guarded by the condition *Pre-agreed SLA about to expire*, from the action state *Create/Update Candidate Service Set* to the action state *Select Service RS for Negotiation*. The remaining validity period threshold that determines when a pre-agreed SLA should be negotiated is selected by the composite service provider.

Following the selection of a service in *RS* for binding at runtime, its SLA is automatically enforced (see the action state *Activate SLA* in Figure 2). When an SLA comes into force, its guarantee terms start being monitored (see the action states *Receive SLA* and *Service & SLA Monitoring* in Figure 2*)*. If the monitoring process detects a violation of the SLA or the deployed service becomes unavailable, the relevant service is replaced by the best available service in *RS* (see the transition from the action state *Service & SLA Monitoring* to the action state *Select & Bind Best Service in RS*). The detection of violation of the conditions of the negotiation triggering rules (e.g. active SLA about to expire) triggers the negotiation phase to establish a new SLA.

The relationship between the quality criteria expressed in a discovery query and the quality preferences expressed in the negotiation rules is exemplified in Figure 3. The figure shows the case where the discovery and negotiation activity take into account two quality criteria, namely Q1 and Q2 where the service consumer (composite service) seeks to maximise the value of Q1 (e.g., service performance) and minimise the value of Q2 (e.g., service cost). The dotted lines *q1* and *q2* in the figure show the minimum acceptable value for Q1 and the maximum acceptable value for Q2 that the service consumer sets, respectively. These two boundary lines should be expressed by quality constraints in the service discovery query to ensure that no service which does not satisfy them will be considered any further and could participate in a negotiation process that is known to fail (such services cannot become members of the set *RS* in Figure 2).

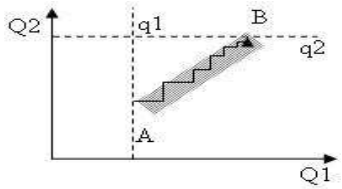


Figure 3. Negotiation rules and query criteria

The rectilinear line in Figure 3 shows the preferences expressed by the negotiation rules of the service consumer. The figure shows a typical case where the consumer is prepared to accept increases in Q2 in exchange of increases in Q1 but cannot agree on any Q2 value that is higher than q2 or any Q1 value that is lower than q1. Thus, the negotiation process will only try to agree an SLA with Q1 and Q2 values in the shaded region of the figure. In this process we assume a multiphase negotiation protocol where participants are allowed to generate counter offers to a given offer until an acceptable goal is reached [13].

4. Specification languages

In this section we describe the languages that are used in our framework to express: (i) service discovery queries, (ii) service level agreements, (iii) SLA negotiation rules, and (iv) SLA negotiation triggering rules.

4.1. Service discovery query language

Service discovery queries are specified in the XML language introduced in [15]. Figure 4 shows an example of a query expressed in this language. The query contains a *StructuralQuery*, *BehaviourQuery* and one or more *Constraints.* The *StructuralQuery* specifies the required interface of candidate services in WSDL. The *BehaviourQuery* specifies the required behavioural characteristics of a service. These characteristics are expressed in a temporal logic language which allows the specifications of conditions about: (a) the existence of certain operations in a service specification; (b) the order in which these operations should be executed by a service; (c) other dependencies between operations; (d) pre-

conditions; and (e) loops concerning execution of certain operations. As they are not related to the negotiation process, the structural and behavioural parts of a query are not further shown in Figure 4. Examples of such parts can, however, be found in [15].

```
<dqns:ServiceQuery xmlns:dqns=
  "http://scube.eu/schema/DiscoveryQuery"
  xmlns:slac="http://scube.eu/schema/Constraint"

xmlns:bqns="http://scube.eu/schema/Behavour_SQL"
   queryID="Q1">
 <dqns:StructuralQuery><!—WSDL->
 </dqns:StructuralQuery>
 <bqns:BehaviourQuery>..</bqns:BehaviourQuery>
 <slac:Constraint>
   <slac:LogicalExpression>
     <slac:Condition relation="GREATER-THAN">
        <slac:Arg1><slac:QualityAttribute
                          name="AVAILABILITY"/>
        </slac:Arg1>
        <slac:Arg2>
           <slac:Constant type="NUMERICAL"
              unit="PC"> 75</slac:Constant>
        </slac:Arg2>
      </slac:Condition>
      <slac:LogicalOperator>AND
      </slac:LogicalOperator>
      <slac:Condition relation="LESS-THAN">
        <slac:Arg1><slac:QualityAttribute
                   name="RESPONSE_TIME"/>
        </slac:Arg1>
        <slac:Arg2>
             <slac:Constant type="NUMERICAL"
                  unit="MS">10</slac:Constant>
        </slac:Arg2>
      </slac:Condition>
    </slac:LogicalExpression>
  </slac:Constraint>
</dqns:ServiceQuery>
```

Figure 4. Example service discovery query

The *Constraint* part of a discovery query comprises a set of constraints specifying the required *QoS* characteristics of a service. The example query of Figure 4 includes a constraint expressed as a logical combination of two conditions. These are: (a) a condition stating that the availability of acceptable services should be greater than 75% and (b) a condition stating that the response time of acceptable services should be less than 10ms. As discussed in Section 3, these constraints will be used to find suitable candidate services during the discovery process and set the bottom line for the required *QoS* characteristics of services during the negotiation process.

4.2. Service level agreement specification

Service Level Agreements (SLA) in our framework are expressed in an XML language based on the query language discussed in 015]. The full XML schema that defines the SLA specification language of our framework is available in 0. Our SLA specification language is used to specify not only final SLAs after an agreement has been reached but also SLA offers (and counter-offers) that different parties may create during the negotiation process.

The attribute *scope* in the *SLAContract* element in this example signifies the status (e.g. under negotiation, pre-agreed SLA or active SLA) of the SLA. As shown in the figure, the SLA contract contains two sets of SLA terms (i.e., constraints over *QoS* attributes). Each set of SLA terms is proposed by a participating actor in the negotiation process. In the example, the first set of SLA terms is proposed by the actor “XYZ” (see the attribute *name* of the sub-element *Company* of the element *Actor*). This actor has the role of a service provider in our example as indicated by the element *Role* in the specification of XYZ.

An actor can also specify its own *negotiation strategy*, i.e., the negotiation protocol that it will use to govern the negotiation process and the communication with the other party during it. Whilst the details of the negotiation strategy are hidden from the other participant, information of the overall protocol that an actor will use should be revealed in order for the two parties to be able to establish whether they are using compatible protocols and it is, therefore, worth engaging in the negotiation process. For the actor XYZ in the example of Figure 5, the negotiation strategy is specified as *MULTI-PHASE_MULTI-ISSUE*. This strategy indicates that XYZ will consider, in principle, counter offers in response to a given offer that it has made until an acceptable goal is reached (*MULTI-PHASE*) and that more than one issue can be the subject of negotiation (*MULTI-ISSUE*).

The SLA required by XYZ in Figure 5 is specified as a logical combination of two conditions. The first of these conditions states that availability of the service offered by the actor is 80%. The second condition states that the response time of the service offered by the actor is 9 milliseconds. Based on this offer, it is clear that XYZ fulfills the boundary conditions of the discovery query of Figure 4 (i.e., AVAILABILITY > 75% and RESPONSE_TIME < 10ms), and it could, therefore, become party to negotiation process where the offer and counter offer of Figure 5 could be generated.

The second set of SLA terms in the example of Figure 5 is proposed by an actor, called “City” which has the *role* of a service *consumer*. Hence, *City* is the service consumer in our example. Furthermore, the service consumer specifies its quality requirements in terms of a set of constraints where each constraint in the set signifies a desired SLA guarantee term.

In this example, the service consumer specifies a constraint that is a logical combination of two conditions: (a) a condition stating that availability should be greater than 90% and (b) a condition stating that the response time should be less than 8ms. It should be noted, that the requestor, in this example has made counter offer

```
<sla:SLAContract xmlns:sla=
   "http://scube.eu/schema/SLA_Contract"
  xmlns:slac="http://scube.eu/schema/Constraint"
  contractID="SLA-No-2" name="S-Cube-SLA"
  scope="UNDER_NEGOTIATION">
 <sla:SLATerms>
   <sla:Actor>
    <sla:Role>PROVIDER</sla:Role>
    <sla:Type>
     <sla:Company name="XYZ"
       contactInformation="Street_Address">
     </sla:Company>
    </sla:Type>
    <sla:NegotiationStrategy>
              MULTI-PHASE_MULTI-ISSUE
    </sla:NegotiationStrategy>
    </sla:Actor>
    <slac:Constraint>
     <slac:LogicalExpression>
      <slac:Condition relation="EQUAL-TO">
        <slac:Arg1><slac:QualityAttribute
                    name="AVAILABILITY"/>
        </slac:Arg1>
        <slac:Arg2>
         <slac:Constant type="NUMERICAL"
                  unit="PC">80</slac:Constant>
        </slac:Arg2>
       </slac:Condition>
      </slac:LogicalExpression>
      <slac:LogicalOperator>AND
      </slac:LogicalOperator>
      <slac:LogicalExpression>
       <slac:Condition relation="EQUAL-TO">
        <slac:Arg1><slac:QualityAttribute
                  name="RESPONSE_TIME" />
        </slac:Arg1>
        <slac:Arg2>
         <slac:Constant type="NUMERICAL"
                    unit="MS">9</slac:Constant>
        </slac:Arg2>
       </slac:Condition>
      </slac:LogicalExpression>
    </slac:Constraint>
 </sla:SLATerms>
```

```
<sla:SLATerms>
  <sla:Actor>
    <sla:Role>CONSUMER</sla:Role>
    <sla:Type>
      <sla:Company name="City"
          contactInformation="Northampton_Sqr">
      </sla:Company>
     </sla:Type>
     <sla:NegotiationStrategy>
          MULTI-PHASE_MULTI-ISSUE
     </sla:NegotiationStrategy>

   </sla:Actor>
   <slac:Constraint>
    <slac:LogicalExpression>
      <slac:Condition relation="GREATER-THAN">
       <slac:Arg1><slac:QualityAttribute
                    name="AVAILABILITY"/>
       </slac:Arg1>
       <slac:Arg2>
         <slac:Constant type="NUMERICAL"
                  unit="PC">90</slac:Constant>
       </slac:Arg2>
      </slac:Condition>
    </slac:LogicalExpression>
    <slac:LogicalOperator>AND
    </slac:LogicalOperator>
    <slac:LogicalExpression>
      <slac:Condition relation="LESS-THAN">
        <slac:Arg1><slac:QualityAttribute
                  name="RESPONSE_TIME" />
        </slac:Arg1>
        <slac:Arg2>
           <slac:Constant type="NUMERICAL"
                    unit="MS">8</slac:Constant>
        </slac:Arg2>
       </slac:Condition>
      </slac:LogicalExpression>
  </slac:Constraint>
 </sla:SLATerms>
 <sla:Penalty>... . . .</sla:Penalty>
</sla:SLAContract>
```

Figure 5. Example SLA

for the attributes availability and response time, in response to the offers made by the service provider.

During the negotiation process the SLA contains multiple sets of SLA terms where each set is proposed by a participating actor in the negotiation process. This facilitates a participant in the negotiation process to consider all the offers made by all the participants without storing the offers in local storage. However, after a successful negotiation when an agreement is reached the SLA contains only one set of SLA terms that includes the list of participants that agreed to the constraints, as well as the penalties that will apply if the SLA is violated and the time validity of the agreed (pre-agreed SLA).

4.3. Specification of negotiation rules and negotiation triggering rules

In our framework, negotiation rules and negotiation triggering rules are specified in an XML language. The schema of this language can be found in [16]. A negotiation rule in this language has the generic structure:

IF (condition) THEN (action) ELSE (action)

Conditions in these rules are expressed as atomic conditions over quality attributes of services or logical combinations of atomic conditions. Rule actions can be of three types: (i) *accept* actions which enable the acceptance of the value of one or more attributes in a given SLA offer, (ii) *reject* actions which enable the rejection of the value of one or more QoS attributes in a given SLA offer, and (iii) *set* actions which allow to define a new value or range of values for one or more QoS attribute.

An example of a negotiation rule is shown in Figure 6. This example expresses a rule used by a service consumer. The rule states that if the service availability offered by a provider is 90% and the offered service price is half of the consumer's expected price then the offer should be accepted.

```
<tnsr:NegotiationRule>
 <tnsr:If>
  <tnsr:LogicalExpression>
   <slac:Condition relation="EQUAL-TO">
    <slac:Arg1>
      <slac:QualityAttribute name="AVAILABILITY"
              party="PROVIDER"/>
    </slac:Arg1>
    <slac:Arg2>
      <slac:Constant type="NUMERICAL">90
      </slac:Constant>
    </slac:Arg2>
   </slac:Condition>
   <slac:LogicalOperator>AND
   </slac:LogicalOperator>
   <slac:Condition relation="LESS-THAN">
    <slac:Arg1>
      <slac:QualityAttribute name="PRICE"
                          party="PROVIDER"/>
    </slac:Arg1>
    <slac:Arg2>
      <slac:ArithmeticExpression>
       <slac:ArithmeticOperand>
          <slac:QualityAttribute name="PRICE"
                        party="CONSUMER"/>
      </slac:ArithmeticOperand>
      <slac:ArithmeticOperator>MULTIPLY
      </slac:ArithmeticOperator>
      <slac:ArithmeticOperand>
        <slac:Constant type="NUMERICAL">0.5
        </slac:Constant>
      </slac:ArithmeticOperand>
     </slac:ArithmeticExpression>
    </slac:Arg2>
   </slac:Condition>
  </tnsr:LogicalExpression>
 </tnsr:If>
 <tnsr:Then>
  <tnsr:Action>
    <tnsr:Accept>
      <tnsr:QualityAttribute name="AVAILABILITY"
                  party ="PROVIDER"/>
      <tnsr:QualityAttribute name="PRICE"
                  party ="PROVIDER" />
     </tnsr:Accept>
    </tnsr:Action>
 </tnsr:Then>
</tnsr:NegotiationRule>
```

Figure 6. Example Negotiation Rule

5. Related Work

An agent based framework for SLA management is presented in [9]. In this framework an initiator agent from the service consumer's side and a responder agent from the service provider's side take part in the negotiation process. The responder agent advertises the service level capabilities and the initiator agent fetches these advertisements and initializes the SLA negotiation process. Different stages of SLA life cycle e.g. formation, enforcement and recovery is performed through the autonomous interactions among theses agents. In the case of an SLA violation, the initiator agent may either claim compensation and renegotiate with the service provider or select a new service provider. Provision of compensation in case of violation of SLA is also argued in [1]. This approach claims that the penalty clauses in the SLA should not only specify the monetary penalties or impact on potential future agreements between the parties; rather the penalty clauses should include several other issues such as which countries laws will be applied in case a conflict between the provider and the client arise, the impact of the penalty clauses on the choice of service level objectives.

Runtime renegotiation is suggested in [4, 7, 5, 2, 3] to manage SLA violations. In [2] service level objectives are revised and renegotiated at runtime and the deployed service is adjusted to the newly agreed service level objectives. A similar approach which allows changing service level objectives whilst keeping the existing SLA is described in [5]. In [3] a renegotiation protocol is described that allows the service consumer or service provider to initiate renegotiation while the existing SLA is still in forced. In this protocol either party may initiate the renegotiation due to the changes in the business requirements and after a successful renegotiation the existing SLA is superseded by a new contract.

All of these approaches are reactive in nature, i.e. renegotiation starts only after an existing SLA is violated. The outcome of renegotiation is either a revised set of service level objectives allowing the acceptance of a service from an existing provider or a new SLA for a new service provider terminating the existing SLA. All these approaches either affect the quality of the delivered service or fail to guarantee uninterrupted service. Our proposed framework integrates SLA negotiation with dynamic service discovery and, hence, can provide integrated runtime support for both these key activities which are necessary in order to achieve the runtime operation of service based applications with minimised interruptions.

6. Conclusion and future work

This paper proposes a framework that integrates service discovery with proactive SLA negotiation. The integrated service discovery/SLA negotiation process can be used by service consumers (i.e., composite services and/or service based applications) in order to identify potential alternative services for the constituent services that they currently use. The identification of alternative services is based on various characteristics of published services including structural, behavioural and *QoS* characteristics.

The framework negotiates with each alternative service that is identified by the discovery process a service level agreement over the *QoS* level of the service. The negotiation process is carried out according to a two-phase protocol that may result in a provisionally agreed but not activated SLA or negotiation failure. A provisional

SLA has an expiry date by which it should either be activated or cease to exist. The objective of proactive SLA negotiation is to ensure that a service which could be potentially used by a service consumer will have an established set of provision terms if the need to deploy it arises suddenly at runtime and, therefore, it won't be necessary to engage in a possibly lengthy negotiation process interrupting the operation of the service consumer application. To achieve this, service providers should be willing to devote resources in negotiating SLAs with service consumers that may never become their clients. This will inevitably lead to some waste of resources from the providers' side. However, providers are likely to be willing to undertake this cost due to the potential of gaining new clients or if this is the only way to maintain existing clients.

The presented framework opens a wide scope for future investigations. The framework can be extended, for example, to support proactive negotiation for hierarchical SLAs, i.e., SLAs of complex composite services deploying other composite services with their own sub-SLAs which will need to be negotiated separately to come to an outermost level agreement. Also in the presented framework, negotiation rules are specified by the participating parties before the negotiation starts and are followed in the negotiation process. The framework can also be extended to support dynamic adaptation of negotiation rules, i.e. scenarios where the participants will be able to dynamically change the negotiation rules during the negotiation process if they realize that they cannot achieve the desired agreements. Finally, a thorough experimental evaluation will be necessary to establish the exact costs and benefits of proactive negotiation under different operational circumstances.

7. Acknowledgement

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under the Grant Agreement 215483 (S-Cube).

8. References

[1] Omer Rana, Martin Warnier, Thomas B. Quillinan, Fances Brazier and Dana Cojocarasu, "Managing Violations in Service Level Agreements", Proc. of the Usage of Service Level Agreements in Grids Workshop, 2007

[2] Di Modica G.,Tomarhio O. and Lorenzo V., "A framework for the management of dynamic SLAs in composite service scenarios", ICSOC 2007, International Workshops, 2007

[3] Michael Parkin, Peer Hasselmeyer, Bastian Koller, Philipp Wieder: An SLA Re-negotiation Protocol. In proceedings of the 2nd Workshop on Non Functional Properties and Service Level Agreements in Service Oriented Computing, November 2008.

[4] Waeldrich, O., Ziegler, W., "A WS-Agreement based Negotiation Protocol", Technical Report, Fraunhofer Institute SCAI, 2006.

[5] Rizos Sakellariou and Viktor Yarmolenko, "On the Flexibility of WS-Agreement for Job Submission", Proc. of the 3rd Int. Workshop on Middleware for Grid Computing, 2005.

[6] Catalin L. Dumitrescu and Ian Foster, "GRUBER: A Grid Resource Usage SLA Broker", Int. Euro-Par conference, 2005.

[7] Philipp Wieder, Jan Seidel, Oliver Wäldrich, Wolfgang Ziegler, and Ramin Yahyapour, "Using SLA for Resource Management and Scheduling - A Survey", *Grid Middleware and Services Challenges and Solutions*, Springer, 2008.

[8] Raimondi, F. and Skene, J. and Chen, L. and Emmerich, W., "Efficient monitoring of web service SLAs", Technical report. Research Notes (RN/07/01). UCL, London, UK. 2007.

[9] Q. He, J. Yan, R. Kowalczyk, H. Jin, Y. Yang, "Lifetime Service Level Agreement Management with Autonomous Agents for Services Provision". Information Sciences, Elsevier, 2009.

[10] Peer Hasselmeyer, Changtao Qu, Lutz Schubert, Bastian Koller, and Philipp Wieder. Towards Autonomous Brokered SLA Negotiation. In: Proceedings of the eChallenges e-2006 Conference, Barcelona, Spain, October 2006.

[11] Paul Karaenke and Stefan Kirn, "Service Level Agreements: Evaluation from a Business Application Perspective", Proceedings of eChallenges 2007

[12] Pichot, A.; Wieder, P.; Ziegler, W.; Wäldrich, O. "Dynamic SLA-negotiation based on WS-Agreement", CoreGRID - Network of Excellence, 2007, Technical Report; 82, TR-0082

[13] V. Robu, D.J.A. Somefun, and J. A. La Poutre. "Modeling complex multi-issue negotiations using utility graphs", In Proc. of the 4th Int. Conf. on Autonomous Agents & Multi Agent Systems (AAMAS'05), Utrecht, ACM Press, 2005

[14] Kyriakos Kritikos and Barbara Pernici, editors. "Initial Concepts for Specifying End-to-End Quality Characteristics and Negotiating SLAs". S-Cube project deliverable, June 2009. S-Cube project deliverable: CD-JRA-1.3.3. http://www.s-cube-network.eu/achievements-results/s-cube-deliverables.

[15] A. Zisman, G. Spanoudakis, and J. Dooley. A Framework for Dynamic Service Discovery, 23rd Int. IEEE/ACM Conf. on Automated Software Engineering, 2008.

[16] SLA Specifications:
http://www.soi.city.ac.uk/~am697/sla/SLA-Specification.zip

Evolving Services from a Contractual Perspective*

Vasilios Andrikopoulos[1], Salima Benbernou[2], and Mike P. Papazoglou[1]

[1] INFOLAB, Tilburg University, Netherlands
[2] LIRIS, Université de Lyon 1, France
{v.andrikopoulos, mikep}@uvt.nl, sbenbern@liris.univ-lyon1.fr

Abstract. In an environment of constant change, driven by competition and innovation, a service can rarely remain stable - especially when it depends on other services to fulfill its functionality. However, uncontrolled changes can easily break the existing relationships between a service and its environment (its customers and providers). In this paper we present an approach that allows for the controlled evolution of a service by leveraging the loosely-coupled nature of the SOA paradigm. More specifically, we formalize the notion of contracts between interacting services that enable their independent evolution and we investigate under which criteria can changes to a contract-bound service, or even to the contract itself, be transparent to the environment of the service.
Keywords: service evolution, service contracts, compatibility, contract invariance, contract evolution

1 Introduction

A number of serious challenges like mergers and acquisitions, outsourcing possibilities, rapid growth, regulatory compliance needs, and intense competitive pressures require changes at the enterprise level and lead to a continuous business process redesign and improvement effort. Service changes that are required by this effort however must be applied in a controlled fashion so as to minimize inconsistencies and disruptions by guaranteeing seamless interoperation of business processes that may cross enterprise boundaries.

In general, we can classify service changes depending on their direct and side effects [1] in *shallow*, where the change effects are localized to the service or are strictly restricted to the clients of that service, and *deep*, that are cascading types of changes which extend beyond the clients of a service, and possibly to its entire value-chain, i.e., to clients of the service clients such as outsourcers or suppliers. Shallow changes characterize both singular services and business processes and require a structured approach and robust versioning strategy to support multiple versions of services and business protocols. Deep changes on

* The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

the other hand are more intricate and require the assistance of a *change-oriented service life cycle* where the objective is to allow services to predict and respond appropriately to changes as they occur [1]. Due to the complexity and scope of deep changes, this paper discusses only shallow changes, and more specifically changes to the Structural layer elements of the Service Specification Reference Model introduced in [2], i.e., to the message content, operations, interfaces, and message exchange patterns (MEPs), roughly corresponding to WSDL artifacts.

The setting discussed has a number of similarities with the fields of *evolution transparency* and *interoperability preservation* that have been discussed in different forms in [3] and [4] (among others), and essentially boil down to preventing incompatibility between interoperating (interacting) services. These works though depend mainly on adaptation mechanisms to maintain interoperability, and adaptation approaches are by definition *a posteriori* interventions focusing on incompatibility identification and resolution by modification of a service. In that sense, the adaptation process can not discern between shallow and deep changes and is unable to prevent the propagation of changes throughout the value chain, since the modification of a service may have unforeseen consequences to the parties that interact with it. For that reason we are focusing on identifying under which conditions changes to a service are shallow and discuss an *a priori* approach that aims to prevent or at least predict and confine the necessity for adaptation.

The goal of this work is therefore to allow the *independent* evolution of loosely coupled interacting parties in a *transparent* manner so as to preserve their interoperability. In this context, the parties involved in an interaction can either be services, or services and client (service-based) applications. We only consider bilateral interactions, and for each such interaction we distinguish two roles: that of the *producer* and that of the *consumer*. It must be kept under consideration that the role of a service, unlike that of an application that always acts as a consumer, can vary depending on the interaction. An aggregate service for example plays both roles: that of the producer for its clients, and that of the consumer when it interacts with the aggregated services to compose a result. To achieve meaningful interoperability in this context, service clients and providers must come to a mutual agreement, a *contract* of sorts between them [5]. A contract of this type formalizes the details of a service in a way that meets the mutual understanding and expectations of both service provider and service client. Building around this idea, we are presenting mechanisms to effectively deal with the evolution of the structural aspect of both parties, while preserving interoperability despite the changes that may affect them. After we lay down this foundation we discuss the evolution of interactions and contracts themselves.

The rest of the paper is organized as follows: section 2 presents a notation for service description that leverages the decoupling of service providers and clients through the introduction of the contract construct (section 3). Section 4 shows how the introduced notions can be used to control the evolution of the interacting parties while maintaining a high degree of flexibility. Section 5 will briefly present related works, and section 6 discusses conclusions and future

work. To facilitate the conversation, we are using the simple service described below as a point of reference:

Example 1 (Running Example). Let's assume the case of a very simple inventory service that checks for the availability of an item and responds that either the purchase order can be fulfilled, or issues a fault stating that the order cannot be completed. The WSDL file of this service is shown in listing 1.

```
...
  <types>
    <xsd:schema targetNamespace="http://e-grocery.com/InventoryService">
      <xsd:complexType name="inventoryItem">
        <xsd:sequence>
          <xsd:element name="orderID" type="xsd:string"/>
          <xsd:element name="itemID" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>
  <message name="InventoryRequest">
    <part name="inventoryItem" type="tns:inventoryItem"/>
  </message>
  <message name="InventoryConfirmation">
    <part name="confirmationMessage" type="xsd:string"/>
  </message>
  <message name="InventoryFault">
    <part name="faultMessage" type="xsd:string"/>
  </message>
  <portType name="InventoryServicePortType">
    <operation name="checkInventory">
      <input name="item" message="tns:InventoryRequest"/>
      <output name="confirmation" message="tns:InventoryConfirmation"/>
      <fault name="fault" message="tns:InventoryFault"/>
     </operation>
  </portType>
...
```

Listing 1: Inventory Service WSDL specification

2 Service Specifications

The WSDL description of the inventory service in listing 1 is far from complete in describing the structural aspect of the service. In specific, apart from providing an unambiguous schema for the service interfaces (the *signature* of the service) to be used by its clients, it lacks completely in providing *a)* any information on the services used by the service itself to fulfill its functionality (if any), and *b)*

the means to connect the information *required* and *provided* by its signatures with that of the signatures of the other services it is using. It is therefore not suitable for describing the interaction of the service with its environment and has to be replaced by a declarative specification that fulfills this role. [2] provides a more exhaustive discussion on the structure and content of such a service specification scheme. For the purposes of this work, we will only define the following constructs:

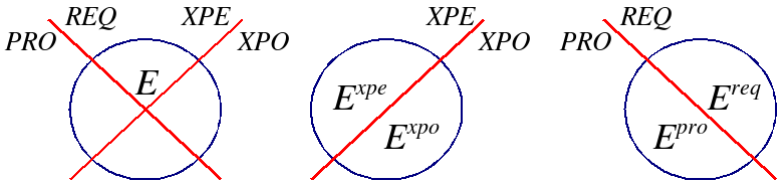
Definition 1 (Element). *An* element *e of a service s is defined as a tuple $(a_1, a_2, \ldots, a_n)$, the set of attributes that characterize the element. a_i is either an atomic attribute or another element e_i of the service.*

For example, `InventoryRequest`, `checkInventory`, and the rest of the WSDL constructs in listing 1 can be represented as elements a_1 =`(inventoryItem)`, a_2 =`(item, confirmation, fault)`, etc.

Definition 2 (Subtyping). *The* specification *E of a service is defined by the set $E = \{e_i, i \geq 1\}$ of its elements. We associate to E the reflexive and transitive relation* subtyping $\leq$ *on elements $(E, \leq)$ defined as: $e \leq e' \Leftrightarrow \{a_1, \ldots, a_n\} \subseteq \{a'_1, \ldots, a'_m\}, m \geq n \wedge a_i \leq a'_j, 1 \leq i \leq n, 1 \leq j \leq m$.*

As discussed in the previous section, the approach discussed by this work assumes that *a)* both producer and consumer are in the general case services, and therefore use the same notation to describe their specifications, and *b)* a producer in one interaction can also act as the consumer for another interaction. This latter interaction may or may not be related to the producer's function in the former. Beyond this generic case, the same paradigm can also be applied to "simpler" cases: autonomous services that implement all of their offered functionalities without using other services act only as producers. Non-service clients (e.g., GUI-supported applications) can be perceived as special cases of exclusive consumers.

We define two orthogonal *views* on E (see figure 1a): the expositions/expectations view and the required/provided view:



(a) Both views (b) XPO/XPE view (c) REQ/PRO view

Fig. 1: Views on the service specification

These views provide us with reference points in disambiguating the roles and functions of elements in a service specification like listing 1. More specifically:

2.1 Exposition/Expectation view

This view (figure 1b) classifies the elements within a service specification with respect to whether they are offered as an interface to the environment or they are "imported" into the service specification, referring to interface elements of other services. In the former case, the service acts as a producer; in the latter as a consumer. Elements of a service specification can therefore fall into one of the following categories:

- Exposition E^{xpo}: the set of elements that describe the offered functionality of the service.
- Expectation E^{xpe}: the set of elements describing the perceived offering of functionality to the service by other services.

The WSDL file of the inventory service for example in listing 1 contains the information on how to access the elements that constitute the inventory service and what information is exchanged while accessing it. From the perspective of the producer of the service, this file specifies what the producer will offer to the service customers: if the `checkInventory` operation is invoked using the `InventoryServicePortType` and the message payload defined, the generated result or fault message will be a simple string. The elements of the file are in that sense in the exposition subset of the service producer specification.

On the other hand, when a consumer of this service builds and/or uses an application that incorporates an invocation of this service, the consumer refers to what it *perceives* to be a set of elements that allow it to access the service. To put it simply, the client is built on the premise of a particular specification of the provided interface, being bound for example to the service of listing 1. These elements are therefore contained in the expectation subset of the consumer specification. What becomes apparent from this is that the same elements can either be expositions or expectations; it only depends on the adopted viewpoint.

Ideally, this perceived specification and the actual specification of the provided service are the same - and that is so far the fundamental assumption in service interactions. But changes to either side, as we will discuss in the following sections, could lead to inconsistencies - in other terms incompatibilities - between those two.

2.2 Required/Provided view

The division enforced by this view (figure 1c) is much more straightforward: it provides the means to cleanly separate input from output in a service specification (irrespective of if it acts as a producer or a consumer). More specifically:

- Required E^{req}: contains the input-type elements of the service specification.
- Provided E^{pro}: contains the output-type elements.

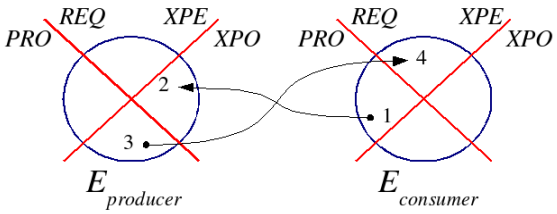
`InventoryRequest` for example is clearly a required element for the producer: it is the input message type for the service. At the same time it is a provided

element for the consumer since it has to be provided to the producer in order to use the respective operation. `InventoryConfirmation` and `InventoryFault` are respectively provided elements for the producer - they are produced as output by the service in one way (normal result) or another (fault message) - and required elements for the consumer (input to it).

2.3 Combining the views

Since the two views are orthogonal, they can be used in conjunction to describe the elements of a service specification: $E^{xpo} \cup E^{xpe} = E^{req} \cup E^{pro} = E$ (figure 1a).

Example 2. Figure 2 shows how an invocation of the inventory service of listing 1 can be described using the classification presented. Due to the request-response messaging pattern of the `checkInventory` operation, the interaction between the service and its client is broken down into two phases: in the first phase, the consumer (client) is using the expectation element *(1)* to invoke the exposition element *(2)* of the producer (service). Since *(1)* is an output for the consumer it belongs to the $E^{pro}_{consumer}$ set, and *(2)* is in the $E^{req}_{producer}$ as the input of the service. The situation is inversed for the second phase, where the producer uses *(3)* to call back *(4)* in the consumer side.



	Element
1	Consumer:`InventoryRequest`
2	Producer:`InventoryRequest`
3	Producer:`InventoryResponse`
4	Consumer:`InventoryResponse`

Fig. 2: Service Interaction

3 Contracts

This section builds on the notation and classification presented in the previous section to discuss the interaction of parties in a loosely-coupled environment and introduce the notion of *contracts* as the means to leverage the decoupling between producer and consumer.

By the term contract we do not refer to the legal documents that describe a binding agreement, but we use the term in the same manner as the (software) contracts in the Eiffel language [6]. The contracts in this context are documents that record the *benefits* expected by each party from their interaction, and the *obligations* that each party is prepared to carry out in order to obtain these

promised benefits. In that sense, the contract protects both sides by clearly defining what is the acceptable contribution and result for a task described by the contract. Our approach applies the same paradigm on services specifications, using the different views discussed above to distinguish between those benefits and obligations, depending on the role that the service plays.

In specific, there is an important distinction in the way that the producer and the consumer of a service are perceiving a service specification document: the producer promises to offer the service in the manner specified in it (the expositions set), and the consumer accepts this promise and builds a client for it based on this promise (the expectations set). In most contemporary SOA implementations, by using for example Web services technologies, this fundamental difference is bridged by accepting one perspective, that of the producer, and shifting the consumer side perspective accordingly. But in that case the consumer has to adopt any changes and assumptions that are done by the producer. Failure to comply with the producer means that the consumer is unable to use the offered functionality, which explains why producer updates typically fail on the client side.

In order to amend this situation, we propose to use a construct (the contract) that bridges the two perspectives and allows for mapping from and to it by either party. This contract is nothing more than an intermediary specification, containing a set of commonly agreed elements specified in a party-independent way. By providing a neutral mapping procedure from each party to the contract we minimize the producer/consumer coupling. Furthermore, given a contract, we allow for reasoning by each party *in isolation*, enforcing the separation of concerns and responsibilities in service design and operation. In the following we formally define the contract construct and describe how to formulate a contract between two parties.

3.1 Contract Definition

In principle only a part of the offered service functionalities may be used by a specific client; on the other hand, a client may depend on a number of disparate services in order to achieve its goals. Thus we need a way to identify and isolate the parts of the interacting parties that actually contribute to the interaction. For that purpose, we will denote with $P \subseteq E^{xpo}_{producer}$ and $C \subseteq E^{xpe}_{consumer}$ the subsets from the producer and consumer specifications respectively that participate in the interaction.

Following on, we define a binding function ϑ that reasons *horizontally* between the elements of parties P and C:

Definition 3 (Service Matching). *A service matching is a binding function defined as* $\vartheta : P \times C \rightarrow U, U = P \cup C$ *such that*

$$\vartheta(x, y) = \{z \in U / \begin{cases} x \leq z \leq y, x \in P^{req}, y \in C^{pro} \\ y \leq z \leq x, x \in P^{pro}, y \in C^{req} \end{cases} \} \tag{1}$$

Binding function ϑ is acting in the same manner as a *schema matching* function would. Schema matching aims at identifying semantic correspondences between elements of two schemas, e.g., database schemas, ontologies, and XML message formats [7]. It is necessary in many database applications, such as integration of web data sources, data warehouse loading and XML message mapping. In most systems, schema matching is manual or semi-automatic; a time-consuming, tedious, and error-prone process which becomes increasingly impractical with a higher number of schemas and data sources to be dealt with. In our case though, the matching function relies on the sub-typing relation to automatically identify elements on either party that are semantically related to each other according to their respective schemata.

Example 3. Let's assume that P contains the elements of listing 1 and let's denote by $x \in P^{req}$ the `InventoryItem` element: $x = (a_1, a_2)$, $a_1 =$ `orderID` and $a_2 =$ `itemID`. A consumer of this service that is bound to listing 1 refers of course to the *same* element `InventoryItem`, and as such it holds that $\exists y \in C^{pro}/y = x \Rightarrow \vartheta(x, y) = z = (a_1, a_2)$.

Now consider the case of another consumer that is bound to listing 2 that is exactly the same as listing 1 in all aspects except from the definition of `InventoryItem` which has an extra argument: $y' = (a_1, a_2, a_3)$, $a_3 =$ `comment` to allow for attaching notes to items. As long as the producer can ignore this extra argument in the requests of the consumer, then by its definition $\vartheta(x, y') = z'$ returns two possible values: $z' = (a_1, a_2)$ or $z' = (a_1, a_2, a_3)$; selection of one of these options is a matter of policy in contract formulation (see following section).

```
...
    <xsd:complexType name="inventoryItem">
      <xsd:sequence>
        <xsd:element name="orderID" type="xsd:string"/>
        <xsd:element name="itemID" type="xsd:string"/>
        <xsd:element name="comment" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
...
```

Listing 2: Alternative inventory item definition

Based on the matching function ϑ we can define the *Contract* R between two parties as a service mapping:

Definition 4 (Service Mapping). *A Service mapping is a* Contract R *defined by a triplet* $< P, C, \Theta >$ *between the two parties that is defined as their image under* ϑ*, i.e.,* $\Theta = \{\vartheta(x, y) | x \in P, y \in C\}$*. The elements that comprise* R *are called the* clauses *of the contract.*

The mapping therefore consists of the results of the binding function for all possible pairs in the producer/consumer sets and is formulated by reasoning vertically through the parties. The contract that is produced by this mapping identifies and represents the mutually agreed specification elements that will be used for the interaction of the parties. Figure 3 demonstrates the relation between P, C, and R graphically.

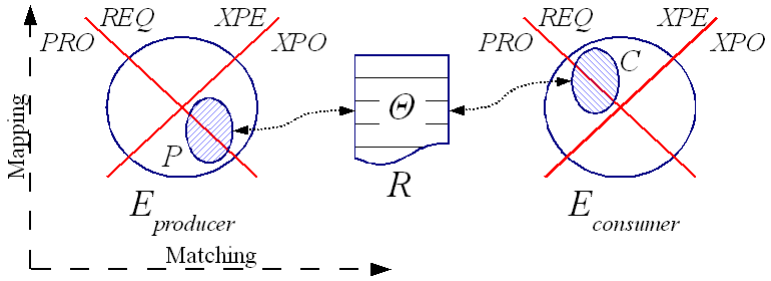


Fig. 3: Producer/Consumer/Contract relation

3.2 Contract Formulation and Management

The definition of contract R between two parties as a service mapping $< P, C, \Theta >$ allows for a straightforward formulation of the contract: given the two parties' specifications P and C, each of which defines the elements through which the interaction is achieved, Θ can be calculated directly by applying the matching function ϑ to them. The issue of contract development therefore shifts in producing P and C from the service provider $E^{xpo}_{producer}$ and client $E^{xpe}_{consumer}$ specifications respectively.

Due to the fact that the service provider is unaware of the internal workings of the service client (represented by the $E^{xpe}_{consumer}$ set) the process of contract formulation is consumer-driven; more specifically, the steps to be followed are:

1. The consumer decides on the functionality offered by the Producer that will be used (if more than one is offered).
2. The set of elements from $E^{xpo}_{producer}$ that fulfill this functionality (e.g., the port type and the associated structural elements) are identified.
3. The identified elements are either copied to the (initially empty) $E^{xpe}_{consumer}$ set or the existing $E^{xpe}_{consumer}$ is used.
4. The image of P and C under ϑ set is calculated. If the resulting set is empty then the image is attempted to be re-calculated using alternative values from ϑ (or cancelled in case all possibilities have been exhausted); otherwise the contract $R =< P, C, \Theta >$ is produced.
5. The consumer submits the formulated contract R to the producer for posterity and begins interaction with producer.

The formulating, storing, and reasoning aspects of the proposed solution can be incorporated in the service governance infrastructure that supports each party. Since ϑ may return one or more possible values, depending on the subtyping 'distance' in the element definition between the producer and consumer specification, a minimum level of 'insight' on the consumer side is required in selecting the appropriate elements from the producer and in assigning values to the binding function ϑ:

Conservative selection policies would opt for the values contributed by the consumer to the calculation of ϑ, trying to protect the consumer from possible changes to the producer.

Liberal selection policies on the other hand would pick the values contributed by the producer and allow for the possibility of the consumer evolving in the future.

The type of policy to be followed is therefore largely a design and governance issue and has to be dealt as such. The solution presented assumes that producers and consumers have the means to formulate, exchange, store, and reason on the basis of contracts. In absence of these facilities from one or both parties the interaction between them reverts to the non contract-based modus operandi that, as we have discussed above, can not guarantee interoperability. The exchange of contracts requires the existence of a dedicated mechanism for this purpose that is not part of the service specification.

4 Contract-Controlled Service Evolution

The previous section discussed how to leverage the loose coupling of the producer and the consumer by means of the contract construct. The following section discuss how this design solution enables evolutionary transparency that preserves (under certain conditions) the producer/consumer interoperability.

In the initial 'static' state of two interoperating parties P and C, and after a contract $R =< P, C, \Theta >$ has been formulated and accepted between them, it holds in general that $P \equiv \Theta \equiv C$. For example, when a simple client is using the service described in listing 1 it is safe to assume that due to the granularity of the service, the client will be using the one (and only) functionality provided by it. That in turn means that it will refer to all the elements contained in the WSDL file. Therefore, $P \equiv C$ and by the definition of the contract construct, $P \equiv \Theta \equiv C$.

But since either party can, or at least should be able to evolve independently of the other, shifts from this state can occur. When changes for example occur to the producer then it may hold that $P' \not\equiv \Theta \equiv C$, or for the consumer side $P \equiv \Theta \not\equiv C'$, or both. These latter states reflect situations of incompatibility between producer and consumer and they have to be prevented from occurring in order to avoid the occurrence of deep changes in the context of the interacting parties. The introduction of a contract between them allows us to reason about the contribution of each party to the interaction without directly affecting the other

party, ensuring that each party is able to evolve *independently* but *transparently*, that is without requiring modifications, to each other.

For that purpose we will distinguish shallow changes occurring to a party in two categories: those that respect the *contractual invariance* and those that require *contractual evolution*. Changes to a party that fall in the former category do not affect the existing contract between the parties. Changes in the latter category require modifications to the contract but nevertheless do not require changes to the other party.

4.1 Contract Invariance

Taking advantage of the ability to reason exclusively on one party given an existing contract, without the need for the other party to participate in this reasoning, exemplifies the notion of independence in evolution. In order to show how this is accomplished we will first formally define what it means for a (modified) party specification to respect, or to be *compliant* with a contract:

Definition 5 (Compliance to Contract). *A version of a party, e.g. version P' of producer P, is said to be* compliant *with respect to an existing contract $R =< P, C, \Theta >$ with a consumer C denoted by $P' \vDash_R C$ iff*

$$\forall z \in \Theta / \exists x' \in P', \vartheta(x', y) = z, y \in C \tag{2}$$

Corollary 1. *Consequently, P'* violates *R, and we write $P' \nvDash_R C$, iff $\exists z \in \Theta / \forall x' \in P', \vartheta(x', y) \neq z, y \in C$.*

The definition above allows for a simple algorithm to check for the compliance of a new version of a party in the producer-consumer relationship: as long as there is a mapping produced by ϑ to *all* clauses of the contract from the elements of the new specification, the two versions are equivalent or *compatible* with respect to the contract - or more formally:

Definition 6 (Compatibility w.r.t. existing Contract).

1. *Given a party, e.g. consumer C, then two versions of the other party, P and P', are called* compatible *w.r.t. a contract R denoted by $P \mapsto_R P'$ iff they are both compliant to R: $P \vDash_R C \wedge P' \vDash_R C$.*
2. *Two versions of a party S and S' are called* fully compatible *iff they are compatible for all contracts $R_i, i \geq 1$ that they participate in, either as producers or consumers: $S \mapsto_{R_i} S' \; \forall R_i$.*

Example 4. Consider the modifications applied to the service specification as depicted in listing 3. Let's assume that it is P that is modified in this way; in that case P' is compatible with P, since they are both compliant to the same contract R. To prove that, we start with the observation that element $x =$`InventoryConfirmation` in listing 1 is in the P^{pro} set, and therefore contributes to the second leg of the binding function (1) which means that

$$\exists y \in C^{req}, z \in \Theta / y \leq z \leq x. \tag{3}$$

Let's denote with x' the changed element from listing 3. It holds that $x \leq x'$ and in conjunction with (3) we get: $\exists y \in C^{req}, z \in R/y \leq z \leq x'$. Thus, $\vartheta(x', y) = \vartheta(x, y)$, and since the rest of the matchings remain unchanged, by (2) we can deduce that $P' \vDash_R C$. If listing 3 though is depicting changes to the customer side, then by the same reasoning we can easily prove that C and C' are not compatible, since $P \nvDash_R C'$.

```
...
<message name="InventoryConfirmation">
  <part name="confirmationMessage" type="xsd:string"/>
  <part name="confirmationDate" type="xsd:date"/>
</message>
...
```

Listing 3: New inventory Service WSDL specification

Figure 4 illustrates the similarities between service matching/mapping and compliance/compatibility: reasoning in the former cases is performed horizontally and in the latter ones vertically.

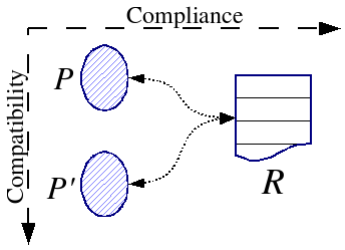


Fig. 4: Compliance vs. Compatibility

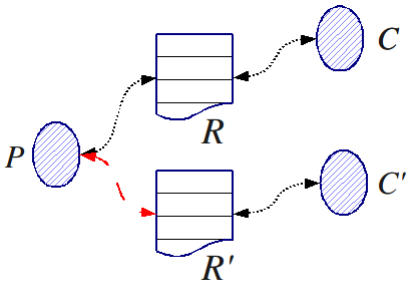


Fig. 5: Backward Compatibility

4.2 Contract Evolution

The previous section discusses the criteria under which changes to one party can leave the contract between them intact, essentially ensuring that these changes are shallow. This does not necessarily mean that all changes that do not respect this criteria are deep. The existing interaction between the parties can be preserved in certain cases, despite the necessity to modify the contract due to changes to one or both of the parties involved, defined as *backward* and *forward* compatibility preserving cases:

Definition 7 (Backward Compatibility). *Two contracts* $R =< P, \Theta, R >$ *and* $R' =< P, \Theta', C' >$ *are called* backward compatible *and we write* $R \mapsto_b R'$ *iff* $\forall x \in P / \exists z' \in \Theta', \exists y' \in C', z' = \vartheta(x, y')$.

In that case (see figure 5), changes to the consumer side leave the producer unaffected. The (new) consumer will use the producer in the same manner as the old consumer did.

Definition 8 (Forward Compatibility). *Two contracts* $R =< P, \Theta, R >$ *and* $R' =< P', \Theta', C >$ *are called* forward compatible *and we write* $R \mapsto_f R'$ *iff* $\forall y \in C / \exists z' \in \Theta', \exists x' \in P', z' = \vartheta(x', y)$.

Similarly, forward compatibility allows for the seamless replacement of the (old) producer with a new producer in the interaction with the consumer without the latter party to have to be modified in any way.

By combining the two definitions we can define when two contracts are compatible:

Definition 9 (Contract Compatibility). *Two contracts* $R =< P, \Theta, R >$ *and* $R' =< P', \Theta', C' >$ *are called compatible and we write* $R \mapsto R'$ *iff they are both backward and forward compatible:* $R \mapsto_b R' \wedge R \mapsto_f R'$.

Contrary to the case of contractual invariance, evolution of the contract itself requires of the parties to exchange a new contract and replace the old contract with the new one. This creates an additional communication overhead that nevertheless has to be weighted against the cost of possible inconsistencies in the current and future interactions of the parties due to the discrepancy between the contract versions.

5 Related Work

The term 'contract' and the approach of introducing contracts in software components design stems from the Eiffel language [6], [8]; the core ideas of that work have greatly influenced our approach.

There are a number of works discussing the introduction of adapters between interacting parties to ensure their interoperability: [9], [10], [3], [11], [12], and [4] among others. Of specific interest to us is the work in [13], since they also make a clear distinction between the service producer and service consumer interfaces and protocols and use mappings to bridge them. Then they proceed to describe how to semi-automatically identify and resolve incompatibilities (mismatches) on interface and protocol level. Our approach extends this idea of separating producer and consumer specifications, but discusses how to avoid mismatches altogether instead of resolving them.

Furthermore, the W3C Technical Architecture Group has published an editorial draft on the extensibility and versioning of XML-based languages [14]. Their findings build on a number of previously developed theories and techniques like [15], [16] and draw lessons from the HTML and HTTP standards. They show

how compatibility can be defined in terms of set theory, using super-sets and subsets to ensure compatibility. Our approach follows a similar way in dealing with the issue of compatibility, but instead of allowing the direct producer/consumer interaction, it introduces the contract construct as an intermediary to further decouple them.

The notion of service mapping comes from the field of schema evolution, i.e., the ability to change deployed schemas - metadata structures formally describing complex artifacts such as databases [17],[18],[7], messages, application programs or workflows. Typical schemas thus include relational or object-oriented (OO) database schemas, conceptual ER or UML models, ontologies, XML schemas, software interfaces and workflow specifications. Effective support for schema evolution is challenging since schema changes may have to be propagated, correctly and efficiently, to instance data, views, applications and other dependent system components. Our approach provides the means to identify schema changes that do not result in propagation of changes.

6 Conclusions & Future Work

In the work presented in the previous sections, we present an approach that allows for transparency in the evolution of a service as viewed from the perspective of both clients and providers, in the context of the loosely-coupled nature of the SOA paradigm. For that purpose we introduce the contract construct as the means to leverage the decoupling of the interacting parties. We present a contract constructing function that bridges the gap between service matching and service mapping. Following on, we build on contractual invariance and contractual evolution to show how to effectively deal with shallow changes to the service provider and client interaction - without the need for adaptation which may lead in turn to deep changes.

There are of course a number of issues that are briefly discussed by our approach that we plan to work on in the future. The matter of management of the contracts and its relationship to service governance mechanisms is the most important issue at hand, since it can provide further insights on the proposed solution. Furthermore, the binding function ϑ value selection policy has to been further investigated. Using a static selection policy can be very restricting; a balancing mechanism for example can be applied for a more dynamic approach, expressed for example by negotiation between the parties in deciding the terms of the contract. Such a negotiation process during the formulation of the contract could result in the offering of additional or more specialized functionalities by the producer and could add a feedback loop to the presented algorithm for contract formulation. A promising direction when it comes to the implementation of our approach is to see whether it is possible to use techniques like the mapping constraints and tools developed by the schema mapping community like ToMAS [7].

The preservation of interoperability enforced by our approach is only the foundation in discussing the evolution of the interaction of parties. Following

on, we plan to investigate how we can build on this work to deal with deep changes and the propagation mechanisms that run through them. On the other hand, another of the limitations of this work, the focus on the structural aspect of the service specification has also to be investigated, and examined if it is possible to apply the same approach to business protocols and policy-related constraints.

References

1. Papazoglou, M.P.: The challenges of service evolution. In Bellahsene, Z., Léonard, M., eds.: CAiSE, Springer (2008) 1–15
2. Andrikopoulos, V., Benbernou, S., Papazoglou, M.P.: Managing the evolution of service specifications. In Bellahsene, Z., Léonard, M., eds.: CAiSE, Springer (2008) 359–374
3. Ponnekanti, S.R., Fox, A.: Interoperability among independently evolving web services, Toronto, Canada, Springer-Verlag New York, Inc (2004) 331–351
4. Senivongse, T.: Enabling flexible cross-version interoperability for distributed services, IEEE Computer Society (1999) 201
5. Papazoglou, M.P.: Web Service: Principles and Technology. Prentice Hall. Addison-Wesley (E) (2007)
6. Meyer, B.: Applying "design by contract". Computer **25** (1992) 40–51
7. Velegrakis, Y., Miller, R.J., Popa, L., Mylopoulos, J.: Tomas: A system for adapting mappings while schemas evolve. In: ICDE. (2004) 862
8. Meyer, B.: Object-Oriented Software Construction (2nd ed.). 2nd edn. Prentice Hall PTR, Upper Saddle River, NJ, USA (1997)
9. Yellin, D.M., Strom, R.E.: Protocol specifications and component adaptors. ACM Trans. Program. Lang. Syst. **19** (1997) 292–333
10. Evans, H., Dickman, P.: Drastic: A runtime architecture for evolving, distributed, persistent systems. Lecture Notes in Computer Science **1241** (1997) 243??
11. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing adapters for web services integration. In: CAiSE, Springer (2005) 415–429
12. Kongdenfha, W., Saint-Paul, R., Benatallah, B., Casati, F.: An aspect-oriented framework for service adaptation. (2006) 15–26
13. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions, Banff, Alberta, Canada, ACM (2007) 993–1002
14. Orchard(edt.), D.: Extending and versioning languages: Terminology (2007) W3C Technical Architecture Group.
15. Orchard, D.: A theory of compatible versions (2006) Published: xml.com article.
16. Hoylen(edt.), S.: Xml schema versioning use cases (2006) Published: W3C XML Schema Working Group Draft.
17. Miller, R.J.: Retrospective on clio: Schema mapping and data exchange in practice. In: Description Logics. (2007)
18. Fuxman, A., Hernández, M.A., Ho, C.T.H., Miller, R.J., Papotti, P., Popa, L.: Nested mappings: Schema mapping reloaded. In: VLDB. (2006) 67–78

Adaptation of Service-based Business Processes by Context-Aware Replanning

Antonio Bucchiarone, Raman Kazhamiakin, Marco Pistore and Heorhi Raik
Fondazione Bruno Kessler
Via Sommarive, 18, Trento TN 38100, Italy
{*bucchiarone,raman,pistore,raik*}*@fbk.eu*

***Abstract*—Business processes are typically used by organizations to meet a specific business goal by executing a set of coordinated activities realized through Web services and service compositions. Operating in open and dynamic environments, business processes often need to be adapted during the execution to react to changes and unexpected problems. The aim of this paper is to provide a dynamic and flexible way to adapt business processes to run-time context changes that impede the achievement of a business goal. We define a formal framework that uses a planning technique to adapt the execution of the service-based business process at runtime in case of context changes. The adaptation enables the business process to continue its normal execution by recovering it to a context, in which the original goal can be achieved. The proposed solution is implemented and validated using a scenario from the logistics domain.**

***Keywords*-Business Processes, Adaptation, Services, Context, Planning.**

I. Introduction

In recent years, service-oriented architectures have been widely used for the realization of complex business processes [1]. Services and service compositions are exploited in a wide range of application domains, such as logistics, supply chain management, user-centered processes, etc. In such processes the activities are realized through the invocation of a set of available services that could be software services but also human-based services, where the human actors realize particular tasks.

Modern business processes often operate in very dynamic, open, and distributed environments, where the relevant properties are changing continuously. Sometimes these changes can compromise the overall execution of the process causing the failure of the entire application execution. In order to react to such failures, there is a need for mechanisms that bring the process execution back to the context from which the business goals can be achieved and the normal execution can proceed. One way to have a business process adaptable is to consider a priori all the cases in which the process can deviate from the normal behavior (i.e., Exception Handling [2]). In this way, it is possible to completely characterize the reaction of the system at design time as the activities (or even subprocesses) implementing the recovery behavior. This specification may be performed by extending standard languages (e.g., BPEL) with the adaptation-specific tools [3], [4], using a set of predefined adaptation rules [5], [6], using aspect-oriented approaches [7]–[9], or modeling and managing business process variants [10]. However, in many situations, such approaches fail to completely solve the problem of adaptation. First, for complex processes the variety of alternatives that require specific recovery actions may be too large for the designers to consider, making the business process hard to define, implement, and revise. And still, unexpected situations and changes still may occur at run-time, requiring the adaptation to be performed even if the concrete case (and its handling) has not been anticipated at design time. To deal with such situations, the adaptation mechanisms should be much more dynamic and flexible, so that the process can recover from critical deviations without defining them a priori.

In this paper, we aim at providing a formal framework that uses an efficient automated planning technique to adapt the execution of the business process at runtime considering the state of the context. This approach relies on the two key ideas. First, adaptation activities are not explicitly represented and are not associated to a concrete context situation or context change. Instead, they are dynamically derived from the currently observed context, the state of a business process, and business goals. To support this, we propose an explicit yet compact and efficient model of context, services, and business goals. Second, we propose an execution environment, where the context of the process is continuously observed, and the adaptation is triggered in case of deviations from what is expected by the business process activities and policies.

The rest of the paper is structured in the following way. In Section II we present our motivating example and discuss the main challenges we face. Section III provides an overall picture of the presented approach, while in Section IV the solution is outlined. Specifically, it includes the formal framework and the implementation of the adaptation mechanism. Section V is devoted to the evaluation of the approach using the motivating example and the prototype implementation of the architecture. In Section VI we discuss related work and conclude the paper with open issues we plan to address in the future.

II. Problem Statement

In this section we present the reference scenario from the logistics domain to illustrate the problem addressed in the paper. In this scenario a business process is implemented to handle the delivery of cars from ships to the retailers in the automobile terminal of the Bremen sea port [11]. Activities of the process include unloading the cars from the ship, storing, applying treatment procedures to meet the customer's requirements and distributing to the retailers. This business process, depicted in Figure 1, is realized as an orchestration of appropriate services, which can be atomic (e.g., Car Check Service, Unloading Service, etc.) or composite (e.g., Store Car Service), they may exhibit non-deterministic behavior, and perform asynchronous message exchanges. Along with the services directly involved in the main process, additional various services are available in the application to deal with specific situations.

The business process described here is defined under an assumption that the involved activities are executed successfully leading to proper changes in the operated domain. However, numerous unexpected situations may take place at runtime. For example:

- *A vehicle may be damaged while moving from the ship to the storage area.* While the necessary activities for storage (e.g., ticket request) have already been activated, according to business policies defined for the domain the damaged cars cannot be stored, and the process fails.
- *There is a queue at the treatment station.* A vehicle has to be brought for treatment (according to specific customer requests) immediately after the registering at the storage area, but the treatment station is completely loaded and the procedure cannot be applied. While normally the availability of places is ensured by the storage service, some of the treatments could not be complete on time leading to the situation not foreseen by the process.

In the situations of these kind the process cannot proceed with normal activity execution and fails. In order to be able to recover business process at run-time, there is a need for appropriate process adaptation mechanisms. While there exist approaches that achieve adaptability by the use of predefined exception handlers, ECA-rules, or explicitly modelled process variants, in our settings they often fail. Indeed, in such a volatile context the set of possible situations to be considered at design time is too large and, moreover, unexpected situations still may take place.

Ultimately, what we need is a framework, which provides:

- *Context-awareness*: the role of the context is fundamental in realizing the adaptation activities [12] as it enables identifying *when* the process adaptation needed and *what* should be done. Specifically, we need 1) to model the context, 2) to relate services to context changes (to understand in which context they can be executed and how they affect the context), and 3) to observe the current process context.
- *Dynamic Adaptation*: to deal with numerous and often unforeseen changes, it is critical to be able to construct adaptation plan dynamically [13], without having all possible situations predefined in advance. In case of damaged car, based on the location of the process and the state of context, it should be possible to construct a new subprocess that involves evacuation service, car repairing, and finally bringing the vehicle to the storage area. In other words, we require dynamic that does not require any involvement of the process designers while executing and adapting the reference process nor needs hard-coded adaptation logics.

In this work we present an approach for dynamic process adaptation, based on automated service composition techniques. While the approach relies on the possibility to observe the business context and its changes, the specific monitoring techniques are out of scope of this paper.

III. Overall Approach

We present a novel adaptation approach that aims at addressing the problems outlined in Section II. The approach relies on the following key elements:

- *Context-aware model of the application*. We explicitly model the business context, in which the process operates, and describe how the services and policies of the application are related to the context and its changes.
- *Context-aware execution framework*. While executing business process, the process context is continuously monitored in order to check whether it changes as expected by the policies and service specifications.
- *Context-aware adaptation based on automated service composition*. If a critical context change is detected, the adaptation tries to recover the process. This is achieved through construction of a composite service that, starting from the actual context state, performs the necessary changes in the domain to bring the process and its context to the expected state.

1) Context model: At design time, along with the definition of a business process and its implementation on top of a set of available services, relevant contextual information is modeled. First, to model the context properties and their evolution, we define *context property diagrams* that capture possible values of the property (as the diagram states) and the changes of the property values (as transitions). Second, to model how the services change the context, we annotate them with the *effects* on the context properties that correspond to changes of the property value (i.e., to some of the transitions in the property diagram). In other words, the context may change due to the service execution or due to some spontaneous, "unexpected" events. Third, we also

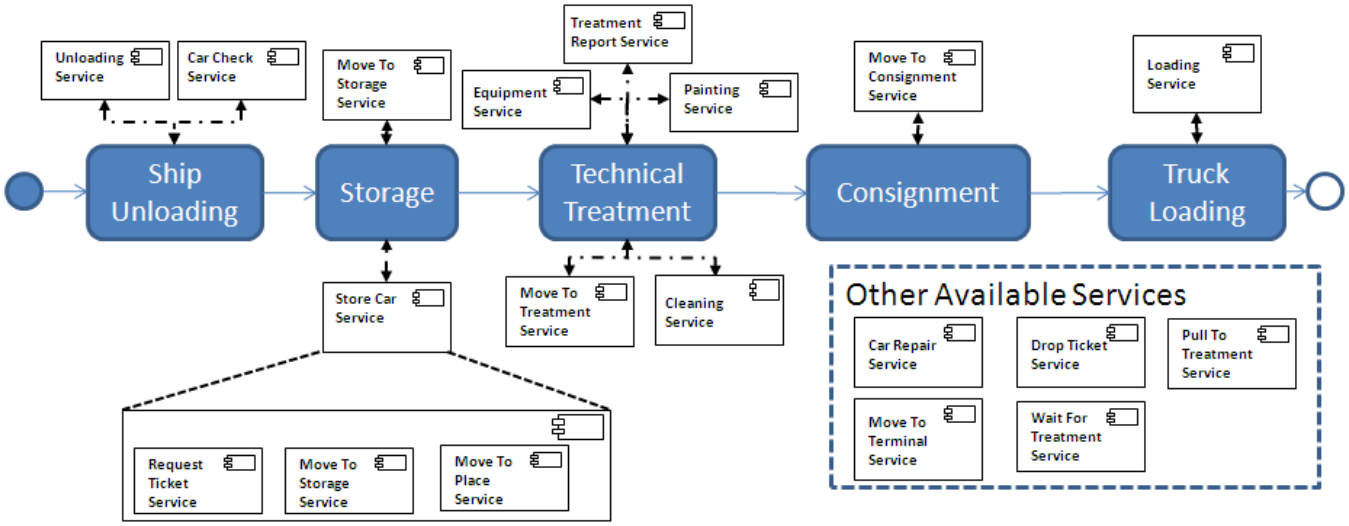


Figure 1: Business Process and Services of the Car Logistics Scenario.

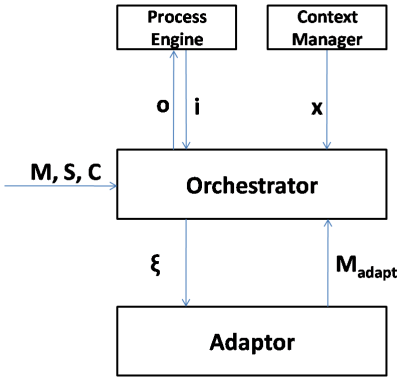


Figure 2: Adaptation Architecture.

capture the business policies over the services to state in which context setting the service may be executed. We do this annotating services with the *preconditions* on the context property values.

2) Execution framework: The reference architecture of our execution framework is depicted in Figure 2. The execution and adaptation of the reference process M is controlled and coordinated by the *Orchestrator* component. The activities of the process are executed by the *Process Engine* component that exchanges messages with the external services. Every activity executed by *Process Engine* is synchronized with and controlled by *Orchestrator*: it receives the execution information o (e.g., messages received, operations performed, etc.) and decides whether to proceed with the next execution step i or to perform process adaptation.

The decision on adaptation is based on the contextual information provided by the *Context Manager* component. *Context Manager* continuously monitors and updates the values of the context properties associated to the process instance[1]. When the observed values of the context properties violates the preconditions of the next activity to execute, the process adaptation is initiated. Based on the current configuration of the context and of the process, *Orchestrator* derives the specific *adaptation problem* ξ and sends the corresponding request to the *Adaptor* component. In response, *Adaptor* generates the new subprocess M_{adapt}, which aims to do its best to "recover", so that the blocked activity can be executed and the main process can continue. *Orchestrator* starts the execution of the generated subprocess and then continues the execution of the main process. If this execution fails again due to exogenous contextual changes, a new round of adaptation is undertaken.

It is important to notice that our adaptation is *completely run-time* and automated. Furthermore, it does not require hard-coded adaptation logics: the specific solution is generated dynamically, based only on the current state and the available services. Another important aspect is that the approach *constantly observes the context and can immediately react to the critical changes*. In particular, even if during the execution of the adaptation process some other problem is detected, the process is immediately terminated and the new adaptation is requested.

3) Adaptation: The adaptation problem ξ sent to *Adaptor* comprises the current status of the system (values of the context properties, state of the involved services), set of available services that may be used for adaptation, and the adaptation requirements. As we already mentioned, the primary objective of the adaptation is to "unblock" the

[1]We remark that the specific techniques for the context monitoring is out of the scope of this paper; approaches like [14] may be exploited for this purpose

process, i.e., to change the context such that the violated precondition of the executed activity is satisfied. To accomplish this, *Adaptor* generates a composition of those services M_{adapt}, which, being executed together, achieve the necessary effect on the context. The construction of the composition is performed with the use of automated planning techniques [15]: the service specifications, the model of context (i.e., context diagrams), and the goal specifications are transformed into the planning problem and the resulting plan is then transformed into the composed service.

It is important to note that for the adaptation we do not consider the whole model of the context and its evolution (which may also be incomplete as some unexpected changes may be unforeseen). Instead, we make an assumption that during the adaptation no exogenous context changes take place, but only those that are entailed by execution of the services involved in the adaptation process. This assumption brings the following advantages. First, we do not necessary need to define at design time all possible evolutions of context: only effects of the services are important. This allows us to react even to completely unexpected changes. Second, this drastically simplifies the planning problem and the construction of the adapting composition becomes very efficient. On the other hand, this assumption does not lead to incorrect results. Indeed, if while the execution of the generated composed process some exogenous contextual change takes place, the *Orchestrator* component will immediately react to this, triggering another adaptation. Given the fact that those exogenous changes aim at representing extraordinary situations and events, such adaptations should also happen rarely, and the recovery activities eventually terminate.

We also remark that as the services may be non-deterministic (e.g., the diagnosis and repair service may complete successfully or may fail if the care is heavily damaged), the primary objective is not guaranteed (e.g., there is no way to proceed with unrepairable vehicle). For such cases, it is possible to associate with the process some "finalizing" recovery goals (i.e., dispose the car) that should become secondary objectives of the adaptation process in such extreme situations.

IV. Solution

In this section we define formally the elements of the presented approach and the solution. In particular, we give the formalization of the business process context and its changes, of the constituent services and their relation to the context, and of the adaptation problem. We then present the solution to the adaptation problem in terms of service composition via planning.

A. The Formal Framework

1) Context Property: In our approach, we explicitly formalize the business context of the reference process as a set of *context properties*. A context property represents some important characteristic of the environment that can change over time. For example, in the logistics scenario context properties may be the location of the car, the status of the car (operable/damaged), the status of the treatment area (busy/occupied), etc. We model the evolution of a context property with a *context property diagram*, which is a state transition system. Here states correspond to possible configurations of a property and transitions stand for possible property evolutions. Each transition is labeled with an event that characterize the change.

Definition IV.1 *(Context Property Diagram) Context property diagram c is a tuple $\langle L, L_0, E, T\rangle$, where:*

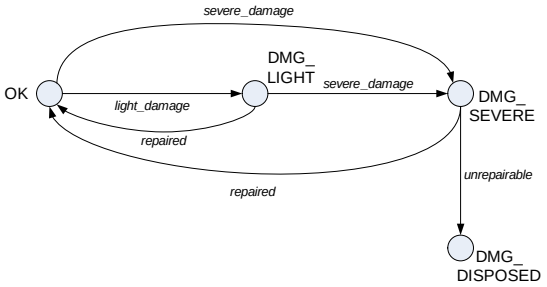
- *L is a set of configurations and $L_0 \subseteq L$ is a set of initial configurations;*
- *E is a set of property-specific events;*
- *$T \subseteq L \times E \times L$ is a transition relation;*

We denote with $L(c)$ and $E(c)$ the corresponding elements of context propert diagram c.

It is important to note that the context property may evolve as an effect of service invocations (e.g., vehicle get repaired after the repair service is engaged), which corresponds to the "normal" behavior of the domain, but also as a result of volatile – "unexpected" – changes. In these regards we can distinguish *controllable* events, i.e., those that may be achieved through services, and *uncontrollable* events, i.e., exogenous events of the environment. With uncontrollable events we capture unexpected situations that may require process adaptation.

Normally, the context is rather complex and consists variety of context properties C. The state of the context is a product of states of its property diagrams.

Example. Context property diagram for the car status may be presented as follows:



The initial state of the diagram is OK (the car is operable). Due to exogenous events, the car can get slightly or severely damaged. The corresponding transitions labeled with events $light\_damaged$ and $severe\_damage$ are presented. A damaged car can be repaired with the repairing service (transitions labeled with event $repaired$). Additionally, a severely damaged car can be recognized as unrepairable with that service (event $unrepairable$).

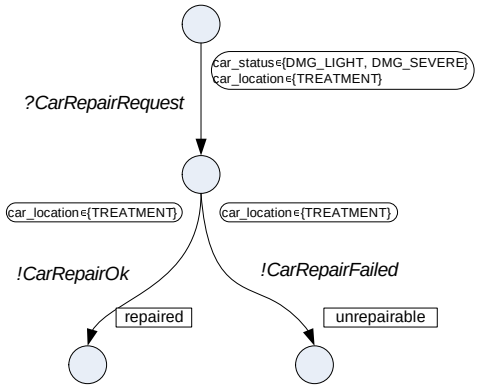
2) Service: In order to model complex service protocols that feature asynchronous, stateful and non-deterministic behavior (e.g., those specified in BPEL), we use state transition systems, where transitions correspond to service operations (input and output messages). To capture the impact of service execution on the domain and to represent the *business policies*, under which we expect services to operate, we annotate service transitions with context *preconditions* and *effects*. A precondition P is a set of configurations of C, in which the execution of the service operation is allowed. We will use these preconditions to detect the need for adaptation. An effect E is a set of property-specific events specified in C, which fire (and thus make C evolve) when a corresponding service transition takes place. As such, effects show how the execution of services affects the context.

Definition IV.2 *(Annotated Service) Annotated service s is a tuple $\langle L, L_0, I, O, T \rangle$, where:*

- *L is a set of states and $L_0 \subseteq L$ is a set of initial states;*
- *I is a set of input actions (receiving a message);*
- *O is a set of output actions (sending a message);*
- *$T \subseteq L \times P^* \times A \times E^* \times L$ is a transition relation, where $A = I \cup O$ is a set of actions; $P = \prod_{c_i \in C} L(c_i)$ is a set of configurations of C, thus P^* stands for a precondition; $E = \bigcup E(c_i), c_i \in C$ is a set of controllable events in C, thus E^* stands for an effect;*

We denote with $L(s)$, $I(s)$ and $O(s)$, the corresponding elements of annotated service s.

Example. The formalization of annotated Car Repair service is represented as follows:



Here, a request (input action *?CarRepairRequest* is followed by a non-deterministic response (output actions *!CarRepairOk* and *!CarRepairFailed*). The preconditions define that the car should be at the treatment area when it is being repaired (precondition $car\_location \in \{TREATMENT\}$) and that the service should be applied only to a broken car (precondition $car\_status \in \{DMG\_LIGHT, DMG\_SEVERE\}$). The effects of the output transitions define that the car may be successfully repaired (event $repaired$, which brings the status diagram to the state OK) or is unrepairable (event $unrepairable$, bringing the property to the state $DMG\_DISPOSED$).

3) Adaptation Problem: The adaptation strategy we adopt in this paper is to recover from the process failure so that the process execution can be resumed from the point where it has been blocked.

Let us consider process M that orchestrates a set of services S, whose preconditions defined over context property diagrams C. When the preconditions of the next activity (e.g., "move a car") to execute are violated (e.g., the precondition is that the car is operable, but the car is damaged), the execution should not proceed and the adaptation of M is required. In order to adapt reference process M we generate adaptation process M_{adapt}, which is the orchestration of services S that implements the above adaptation strategy. The primary goal of resuming the execution of M can be expressed as the reachability of the configurations of services S and context property diagrams C in which the execution of the next activity is possible. As we already mentioned, it is also possible to specify additional configurations to which the process should be brought when the above adaptation cannot be achieved. We remark that those recovery configurations are terminal for the process. We use them as a secondary (i.e., less preferable) goal for our adaptation problem. For example, we may require the car to be disposed, when its repair is not possible anymore.

Formally, adaptation problem may be defined as follows.

Definition IV.3 *(Adaptation Problem)*
Adaptation problem ξ is a tuple $\langle C, S, \mathcal{I}, \mathcal{G}_I, \mathcal{G}_{II} \rangle$, where:

- *C is a set of context property diagrams;*
- *S is a set of services defined over C;*
- *$\mathcal{I}, \in L(s_1) \times \ldots \times L(s_m) \times L(c_1) \times \ldots \times L(c_n), s_i \in S, c_j \in C$ is the current configuration of context property diagrams C and annotated services S;*
- *$\mathcal{G}_I, \mathcal{G}_{II}, \subseteq L(s_1) \times \ldots \times L(s_m) \times L(c_1) \times \ldots \times L(c_n), s_i \in S, c_j \in C$ are sets of primary and secondary goal configurations;*

We denote with $C(\xi)$, $S(\xi)$, $\mathcal{I}(\xi)$, $\mathcal{G}_I(\xi)$, $\mathcal{G}_{II}(\xi)$ the corresponding elements of the adaptation problem ξ.

The solution to adaptation problem ξ is process M_{adapt} that orchestrates services $S(\xi)$. When executed from current configuration $\mathcal{I}(\xi)$, M_{adapt} brings services $S(\xi)$ and context property diagrams C to one of primary goal configurations $\mathcal{G}_I(\xi)$, otherwise to one of secondary goal configurations $\mathcal{G}_{II}(\xi)$. We remark that M_{adapt} succeeds only if no uncontrollable events happen during its execution.

B. Adaptation Strategy and Derivation of an Adaptation Process

In order to automatically solve adaptation problems, we use the variation of the service composition approach presented in [15]. According to it, a composition problem

is transformed into a planning problem and the planning techniques are used to resolve it. So do we.

The planning domain is derived from the adaptive problem. In particular, a set of n services $(s_1, \ldots, s_n)$ and m context property diagrams $(c_1, \ldots, c_m)$ are transformed into STSs using pretty straightforward transformation rules very similar to those presented in [15]. Before this transformation, all transitions labelled with uncontrollable events are removed from context property diagrams (so we plan in "controllable" environment).

So we get STSs $\Sigma_{s_1} \ldots \Sigma_{s_n}$ and $\Sigma_{c_1} \ldots \Sigma_{c_m}$. The planning domain Σ is a product of all STSs of the annotated services and context property diagrams synchronized on preconditions and effects:

$$\Sigma = \Sigma_{s_1} \| \ldots \| \Sigma_{s_n} \parallel \Sigma_{c_1} \| \ldots \| \Sigma_{c_m}$$

Initial state r of the planning domain is derived from the current configuration $\mathcal{I}(\xi)$ of the adaptive system, in which the need for the adaptation aroused.

Finally, the sets of primary and secondary goal configurations (G_I and G_{II} respectively) are transformed into the sets of configurations G_I^Σ and G_{II}^Σ of the planning domain Σ. We denote the planning goal as a reachability goal with preferences:

$$\rho = (G_I^\Sigma, G_{II}^\Sigma)$$

which means that the primary planning goal is to reach one of the states in G_I^Σ, and the secondary goal is to reach one of the states in G_{II}^Σ.

After all, we apply the approach of [16] to domain Σ and planning goal ρ and generate a controller Σ_c (plan), which is such that $\Sigma_c \triangleright \Sigma \models \rho$ (domain Σ reaches goal ρ when controlled by Σ_c). The state transition system Σ_c is further translated into executable process M_{adapt}, which implements the above described adaptation strategy. The back translation from STS into executable specification is quite simple: input actions in Σ_c model an incoming message from a component service, while output actions in Σ_c model an outcoming message to a component service.

Correctness of the approach. The proof of the correctness of the approach consists in showing that, under the aforementioned assumptions, all the executions of the adaptation process M_{adapt} (translation of controller Σ_c) implement the adaptation strategy. Here we outline the key points of the proof. It is easy to see that each execution θ of the adaptation process is also a run of the domain, i. e., if $\theta \in \Pi(\Sigma_c)$ then $\theta \in \Pi(\Sigma)$. Under the requirement that all the executions of the domain terminate in goal states, we get that the executions of the domain implements the adaptation strategy. As a consequence, the following theorem holds.

Theorem IV.1 (Correctness of the approach) *Let:*

- $\Sigma_{s_1}, \ldots, \Sigma_{s_n}$ *be the STS encoding of services* $s_1, \ldots s_n$ *and*
- $\Sigma_{c_1}, \ldots, \Sigma_{c_m}$ *be the STS encoding of context property diagrams* $c_1, \ldots c_m$.

Let Σ_c *be the controller for a particular composition problem*

$$\Sigma = \Sigma_{s_1} \| \ldots \| \Sigma_{s_n} \parallel \Sigma_{c_1} \| \ldots \| \Sigma_{c_m}$$

$$\rho = (G_I^\Sigma, G_{II}^\Sigma)$$

i.e., $\Sigma_c \triangleright \Sigma \models \rho$. *Then the executions* $\Pi(\Sigma_c)$ *implements the adaptation strategy.*

As we mentioned in Section III, we assume that no exogenous context changes may take place during the adaptation, and therefore in the transformation of the adaptive system into a planning domain we ignore all uncontrollable transitions in context property diagrams. Our experiments show that such an approach leads to very efficient adaptation, as it significantly increases the performance of the planning algorithm by reducing the amount of reachable states in the planning domain. Furthermore, in many domains external contextual changes are quite improbable so that additional rounds of adaptation are rarely needed. Compared to [16], we also use preprocessing of the planning domain by pruning unreachable states, which also increases the performance of the planning algorithm.

V. Evaluation

In order to evaluate our approach we implemented the prototype of the architecture presented in Section III and designed the reference scenario using the formal framework introduce in Section IV-A.

In order to formalize the reference scenario with enough details, we distinguished and specified 10 context property diagrams (car location, car status, treatment areas status etc.) and 16 stateful services (services for moving, storing, repairing, treating, checking the car etc.). Service specifications were annotated with preconditions and effects over the context property diagrams. In the initial state the car is on the ship and it is operable. The goal state is where the operable car is properly treated and is loaded on the track waiting for the delivery. The alternative goal state is where the car is unrepairable and at the disposal area. We also specified external context changes that model unexpected light and severe damage of the car and switching of the treatment areas status from "occupied" to "free" and vice versa. Finally an implementation of the reference process was specified.

The implementation of the reference process is quite complex and dozens of different unexpected situations, each requiring specific adaptation, may happen. We will consider only those two mentioned in Section II.

A. Damage on the move

In this example, the car is severely damaged while moving towards the storage area. There is a storage ticket reserved for the car, which guarantees a place at the storage area. This situation requires adaptation since is not considered in the reference process. There is a number of business policies that are implemented in the framework using action preconditions:

- The car can move on its own only if it is not severely damaged;
- The car can be stored only if it is not damaged;
- The car can be stored only if it has a storage ticket;
- The car can be repaired only at the treatment area;
- The car must have no storage ticket when it is being repaired (not to waste the space at the storage area);
- The storage ticket can be received only at the terminal;

The goal of the adaptation is to reach the configuration "the operable car is moving towards the storage with the storage ticket in hands". The recovery goal is to leave the car at the treatment area if it is unrepairable. The adaptation process M_{adapt} obtained (Fig. 3) is completely compliant with the context, business policies and goals (the primary goal is depicted as success and the secondary goal as failure). We remark that the same severe damage in different context settings could result in absolutely different adaptation process generated.

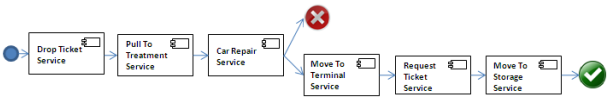


Figure 3: Adaptation process for "Damage on the Move"

B. Waiting for treatment

The car has just moved to the treatment area and is ready to be treated. The treatment consists of three operations: equipment, painting and cleaning. It turns out that the equipment and painting areas are busy. This is an unexpected situation and the adaptation is undertaken. The following policies play role for this case:

- The car should complete all three treatment operations before leaving the treatment;
- Cleaning has to be performed after painting;

The adaptation process (Fig. 4) suggests to use a special notification service that notifies the client about the treatment areas just freed. One can see, that the adaptation is successful only if we receive a message about painting or equipment areas to be free. The cleaning area is not of interest for the moment, since cleaning cannot be performed before painting.

We ran the examples on the double-core 2Ghz Windows XP laptop with 4 Gb of RAM. The derivation of the two adaptation processes took 13 and 3 seconds respectively. While analyzing these numbers we have to realize that although the adaptation processes presented do not feature high complexity, they are derived from quite a large domain that requires significant resolution time. Taking into account the real complexity of the reference scenario, we consider these results as a proof of high potential of our approach, applicable in a vast majority of real application domains.

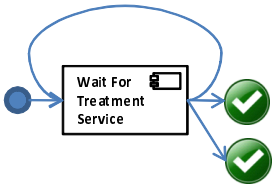


Figure 4: Adaptation process for "Waiting for Treatment"

VI. Related Work and Conclusion

Various frameworks can be found in the literature with the objective to support adaptation of business processes, each of them addressing a specific issue. When the set of possible adaptation configurations are fixed and known a priori, it is possible to completely specify them at design time. In this case, we talk about *built-in adaptation* and the adaptation may be performed using one of the following approaches: by extending standard notations as BPEL with the adaptation-specific tools [3], [4] using a set of predefined adaptation rules [5], [6], [17], using aspect-oriented approaches [8], [9], [18], or modeling and managing business process variants [10], [18], [19]. All the aforementioned approaches are able to capture a precise set of exceptional events or situations and to use for each of them a predefined adaptation rule. In this paper, we have presented a novel approach to adapt business processes where the running application and the adaptation logic are two separate components and the set of possible adaptations are generated directly at runtime when a problem arises (i.e., *dynamic adaptation* [13]). This means that it is not necessary to know at design time which actual conditions will trigger the adaptation, and which kind of adaptation should be performed to correct it. To understand when, how, and why a business process should be adapted we have modeled the execution environment in such a way that its changes are synchronized with the application execution and viceversa. When we need to execute adaptation, this two-way synchronization let the adaptation tool understand what the current state of the business process and the current context are. At the same time, using the current context and its evolution model, we are able to provide the adaptation engine with the precise goal to reach. The adaptation engine generates the adaptation process to execute and brings the main process back to a state from where its execution can be successfully resumed.

A framework similar to our has been proposed in [20]. It is called SmartPM and is able to adapt processes in

case of unforeseen events. However, we consider our formal framework to be much more intuitive and easy to use for the designer. Moreover, the approach in [20] can work with abstract services and does not allow for using real services specified in one of standard languages. On the contrary, our approach is developed for stateful and non-deterministic services that can be modeled using such a language as BPEL and transformed into our formalism automatically [2].

In the future, we plan to extend our framework in order to allow for more sophisticated adaptation strategies that can be chosen automatically. At the same time, our adaptation mechanism has been defined to be applied to single instances of a process model. We want to use a set of adapted instances together with their execution history as training cases for *evolution mechanisms* in order to progressively improve process model that may be further used if a new process instance is to be instantiated.

Acknowledgements.

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube) and from the FP7 EU-FET project 213339 ALLOW.

REFERENCES

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: a Research Roadmap," *Int. J. Cooperative Inf. Syst.*, vol. 17, no. 2, pp. 223–255, 2008.

[2] R. de Lemos and A. B. Romanovsky, "Exception handling in the software lifecycle," *Comput. Syst. Sci. Eng.*, vol. 16, no. 2, pp. 119–133, 2001.

[3] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. P. Buchmann, "Extending BPEL for Run Time Adaptability," in *Proc. EDOC'05*, 2005, pp. 15–26.

[4] A. Marconi, M. Pistore, A. Sirbu, H. Eberle, F. Leymann, and T. Unger, "Enabling Adaptation of Pervasive Flows: Built-in Contextual Adaptation," in *Proc. ICSOC/ServiceWave*, 2009, pp. 445–454.

[5] M. Colombo, E. di Nitto, and M. Mauri, "SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules," in *Proc. ICSOC'06*, 2006, pp. 191–202.

[6] I. Lanese, A. Bucchiarone, and F. Montesi, "A Framework for Rule-based Dynamic Adaptation," in *Proc. TGC 2010*, 2010, pp. 284–300.

[7] A. Charfi and M. Mezini, "AO4BPEL: An Aspect-oriented Extension to BPEL," in *Proc. WWW'07*, 2007, pp. 309–344.

[8] W. Kongdenfha, R. Saint-Paul, B. Benatallah, and F. Casati, "An Aspect-Oriented Framework for Service Adaptation," in *Proc. ICSOC'06*, 2006, pp. 15–26.

[2] http://astroproject.org/

[9] V. Agarwal and P. Jalote, "From Specification to Adaptation: An Integrated QoS-driven Approach for Dynamic Adaptation of Web Service Compositions," in *Proc. ICWS*, 2010, pp. 275–282.

[10] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing variability in business process models: the Provop approach," *Journal of Software Maintenance*, vol. 22, no. 6-7, pp. 519–546, 2010.

[11] F. Böse and J. Piotrowski, "Autonomously controlled storage management in vehicle logistics applications of RFID and mobile computing systems," *International Journal of RT Technologies: Research an Application*, vol. 1, no. 1, pp. 57–76, 2009.

[12] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," in *Proc. HUC*, 1999, pp. 304–307.

[13] A. Bucchiarone, C. Cappiello, E. di Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore, "Design for Adaptation of Service-Based Applications: Main Issues and Requirements," in *Proc. ICSOC/ServiceWave Workshops*, 2009, pp. 467–476.

[14] C. Bettini, D. Maggiorini, and D. Riboni, "Distributed Context Monitoring for the Adaptation of Continuous Services," *World Wide Web*, vol. 10, pp. 503–528, 2007.

[15] P. Bertoli, R. Kazhamiakin, M. Paolucci, M. Pistore, H. Raik, and M. Wagner, "Control Flow Requirements for Automated Service Composition," in *Proc. ICWS'09*, 2009, pp. 17–24.

[16] D. Shaparau, M. Pistore, and P. Traverso, "Contingent planning with goal preference," in *Proc. AAAI'06*, 2006, pp. 927–934.

[17] X. YongLin and W. Jun, "Context-Driven Business Process Adaptation for Ad Hoc Changes," in *Proc. IEEE ICEBE'08*, 2008, pp. 53–60.

[18] G. Hermosillo, L. Seinturier, and L. Duchien, "Using Complex Event Processing for Dynamic Business Process Adaptation," in *Proc. IEEE SCC*, 2010, pp. 466–473.

[19] Z. Jaroucheh, X. Liu, and S. Smith, "Apto: A MDD-Based Generic Framework for Context-Aware Deeply Adaptive Service-Based Processes," in *Proc. ICWS*, 2010, pp. 219–226.

[20] M. de Leoni, "Adaptive Process Management in Highly Dynamic and Pervasive Scenarios," in *Proc. YR-SOC*, 2009, pp. 83–97.

Towards Proactive Adaptation: A Journey along the S-Cube Service Life-Cycle

Andreas Metzger*, Eric Schmieders*, Cinzia Cappiello†, Elisabetta Di Nitto†,
Raman Kazhamiakin‡, Barbara Pernici†, Marco Pistore‡
*Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen, 45127 Essen, Germany
Email: {andreas.metzger, eric.schmieders}@paluno.uni-due.de
‡FBK-Irst
Via Sommarive 18, 38050, Trento, Italy
Email: {raman, pistore}@fbk.eu
†Politecnico di Milano, DEI
Piazza Leonardo da Vinci, 32, 20133 Milano, Italy
Email: {dinitto, cappiell}@elet.polimi.it

***Abstract*—Service-oriented applications are deployed in highly dynamic and distributed settings. Therefore, such applications are often equipped with adaptation capabilities to react to critical issues during their operation, such as failures or unexpected changes of third party services or to context changes. In this paper, we discuss shortcomings of current solutions for adaptive service-oriented applications. To address those shortcomings, we introduce techniques for building and evolving proactive applications. Those techniques have been developed in S-Cube, the European network of Excellence on Software Services and Systems. Proactive adaptation capabilities are considered particularly promising, as they can prevent costly compensation and repair activities. Using those techniques in an integrated way is described along the phases of the service life-cycle. We use a running example to illustrate the shortcomings of current solutions for self-adaptation and to demonstrate the benefits of the S-Cube techniques.**

I. Introduction

Service-orientation is increasingly adopted as a paradigm for building highly dynamic, distributed and adaptive software systems, called service-oriented (or service-based) systems. This paradigm implies a fundamental change to how software is developed, deployed, and maintained [1]: A service-based system cannot be specified and realized completely in advance (i.e, during design-time) due to the incomplete knowledge about the interacting parties (e.g., third party service providers) as well as the system's context and communication infrastructure [2]. Thus, compared to traditional software engineering, much more decisions need to be taken during the operation of the service-oriented system (i.e., after it has been deployed). For instance, those systems will need to react to failures of their constituent services (e.g., if a service provider fails to adhere to its contract) to ensure that they maintain their expected functionality and quality.

In such a dynamic setting, evolution and adaptation methods and tools become key to enable those systems to respond to changing conditions. In accordance with the terminology defined by the S-Cube Network of Excellence [3], this paper differentiates between evolution and adaptation as follows: Evolution is considered as the modification of the system's requirements, specification, models, etc. during design time (also known as maintenance). In contrast, adaptation is considered as the modification of a specific instance of a service-based system during operation. In the current paper we focus on adaptation needed due to some malfunctioning of the system. While the general adaptation due to context changes could also be supported by the proposed techniques, this is not discussed in the present paper.

A. Problem Statement and Related Work

Adaptive systems automatically and dynamically adapt to changing conditions. The aim of adaptation (aka. "self-adaptation") is to reduce the need of human intervention as far as possible. While the behavior of a non-adaptive system is only controlled by user input, adaptive systems consider additional information about the application and its context (e.g., failures of constituent services or different network connectivity). Thus, in order to realize self-adaptive behavior, methods and tools that realize control loops are established that collect details from the application and its context (e.g., by exploiting monitoring mechanisms) and decide and act accordingly [4].

So far, the major work on adaptation has been centered around reactive adaptation capabilities based on monitoring [5]. This means that adaptation is performed *after* a deviation or critical change has occurred. Such a reactive adaptation based on monitoring, however, has at least the following two important shortcomings (cf. [6], [7] and [8]).

- It can take time before problems in a service-based system lead to monitoring events that ultimately trigger the required adaptation. One key trigger for an adaptation should be the case when the service-based system deviates from its requirements (such as expected response time for example). If only those requirements are monitored (e.g., see [9]), the monitoring events might arrive so late that an adaptation of the SBA is not possible

anymore. For instance, the system could have already terminated in an inconsistent state, or the system has already taken more time than required by the expected response time.
- Reactive adaptation can become very costly, especially when compensation or rollback actions need to be performed. As an example, when using stateful (aka. conversational) services [10], the state of the failed service might need to be transferred to an alternative service.

Of course, one can monitor the individual services of an SBA and trigger an adaptation as soon as the service has failed, i.e., violated its contract [11]. However, when using those techniques it remains unclear whether the failure of this service could lead to a violation of the SBA's requirements. This means that there may be situations in which the SBA is adapted although it would not have been necessary, because the requirements might still have been met. Consider the following simple example: Although a service might have shown a slower response time as (contractually) expected, prior service invocations (along the workflow) might have been fast enough to compensate the slower response of that service.

Such unnecessary (or "false positive") adaptations have the following shortcomings [6]:

- Unnecessary adaptations can lead to additional costs and effort that could be avoided. For instance, additional activities such as Service Level Agreement (SLA) negotiation for the alternative services might have to be performed, or the adaptation can lead to a more costly operation of the SBA, e.g., if a seemingly unreliable but cheap service is replaced by a more costly one.
- Unnecessary adaptations could be faulty (e.g., if the new service has bugs), consequently leading to severe problems.

In summary, one key problem that needs to be solved to enable proactive adaptation is to determine whether the service-based application, during its future operation, might deviate from its requirements.

B. Contribution of Paper

This paper describes techniques developed in the S-Cube[1] network of excellence to determine deviations from requirements based on monitored failures. Previous publications (such as [6], [7], [8], [12], [13]) have discussed proactive adaptation techniques mainly in isolation and confined to individual phases of the service life-cycle. A first, more integrated view on adaptation has been presented in [14]. However, the focus was on reactive adaptation and on the design time activities needed to build adaptive service-systems. In contrast, in this paper, we demonstrate how the techniques for determining proactive adaptation play together across the various life-cycle phases and how they can be jointly applied in a meaningful way. As a basis for our discussions, we employ the S-Cube service life-cycle model [15], [14], [16], [17]. In contrast to more traditional life-cycle models, this model considers the specifics of service-based systems, particularly concerning evolution and adaptation.

The remainder of the paper is structured as follows: In Section II, the S-Cube service life-cycle model is introduced. In Section IV, the S-Cube techniques that jointly allow building proactive service-based systems are discussed, differentiating between activities that are done during design-time and activities that are done during the operation phase (run-time). This discussion is illustrated by an example from the eGovernment domain, which is introduced in Section III.

II. The S-Cube Service Life-Cycle Model

The life-cycle models for SBAs that have been presented in the literature (examples include SLDC, RUP for SOA, SOMA, and SOAD, cf. [14] and [18]) are mainly focused on the phases that precede the release of software and, even in the cases in which they focus on the operation phases, they usually do not consider the possibility for SBAs to adapt dynamically to new situations, contexts, requirement needs, service faults, etc.

Specifically, the following aspects have not yet been considered in those life-cycle models:

- *Requirements elicitation and design for adaptation:* The requirements engineering phase includes the elicitation and documentation of the systems functional and quality requirements. In the dynamic setting of SBAs, not only the requirements towards the actual application logic need to be analyzed, designed, and developed, but also the context in which the system is executed needs to be understood [1]. Context changes can necessitate the adaptation of the SBA, for instance if the SLA of a third party service is violated. During design, the capabilities to observe, modify and change the SBA during run-time need to be devised.
- *Extended operation phase:* The operation phase is not only responsible for merely executing and monitoring the application, but it also requires identifying the need for an adaptation of the system as well as the where and how to enact such an adaptation [1].
- *Continuous quality assurance:* Quality assurance has an impact on all aspects of the life-cycle. Therefore, the quality characteristics that are to be assessed and ensured must be identified starting from the requirement analysis phases. Due to open nature, and dynamic contexts in which SBAs operate, quality properties that have a lifelong validity need to be "continuously" asserted [19]. For instance, in the case of third party services, there is no guarantee that a service implementation eventually fulfills the contract promised (e.g., stipulated by an SLA), or it is usually not possible during design-time to model and thus assess the behavior of the underlying distributed infrastructure (such as the Internet).

The service life-cycle model envisioned by the S-Cube network aims at incorporating those aspects. The S-Cube service life-cycle model [15], [14], [16], [17] relies on two development and adaptation loops, which can be executed in an incremental and iterative fashion:

[1] http://www.s-cube-network.eu/

- The *development and evolution loop* (see right hand side of Figure 1) addresses the classical development and deployment life-cycle phases, including requirements and design, construction and operations and management (see Section II-A).
- The *operation and adaptation loop* (see left hand side of Figure 1) extends the classical life-cycle by explicitly defining phases for addressing changes and adaptations during the operation of service-based applications (see Section II-B).

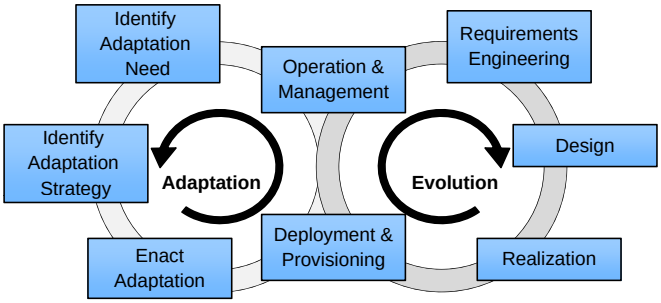


Fig. 1. The S-Cube Service Life-Cycle

A. Development and Evolution Cycle

Requirements Engineering. In the requirements engineering phase, the functional and quality requirements for the SBA are elicited and documented. The specifics of SBAs make the requirements engineering phase particularly relevant. This is related to the highly dynamic nature of SBAs and to the necessity to guarantee the continuous adaptability and the evolvability of these applications. Indeed, in a context where the application is in continuous evolution and is characterized by very blurred boundaries, the study of those requirements that exist a priori in the organizational and business setting, and that are hence largely independent from the solution, becomes very important.

Design. During the design phase, the activities and the control flow of the application are specified. In the service-oriented case, this usually means that a workflow is specified using languages such as BPEL. Together with the definition of the workflow, candidate services are identified that can provide the functionality and quality to fulfill the requirements of the SBA. This means that those services that cover, at least partially, the expected functionality and quality of service are identified. This is supported by service matchmaking techniques, such as the ones presented in [20]. A further task in this phase is to define adaptation strategies and mechanisms which enable the application to react to adaptation needs (cf. [14]).

Construction. After the design phase, the construction of the system can start. Especially, it has to be taken into account that SBAs are obtained by the integration and coordination of services from different providers. Specifically, this means that for establishing the desired end-to-end quality of those SBAs, contracts between the service providers and the service consumers on quality aspects of services have to be established. Typically, this requires some form of SLA negotiation and agreement. Following [20], this means that for each service, the best quality of service level for the available budget is negotiated with the providers of the candidate services that have been identified in the previous phase.

Deployment and Provisioning. The deployment and provisioning phase comprises all the activities needed to make the SBA available to its users. It should be noted that an SBA can itself be offered as a service.

B. Operation and Adaptation Cycle

Operation and Management. This phase specifies all the activities needed for operating and managing an SBA. The literature also uses the term governance to mean all activities that govern the correct execution of SBAs (and their constituent services) by ensuring that they provide the expected functionality and level of quality during operation. In this setting, the identification of problems in the SBA (e.g., failures of constituent services) and of changes in its context play a fundamental role. This identification is obtained by means of monitoring mechanism and, more generally, by exploiting techniques for run-time quality assurance (such as online testing or run-time verification). Together, those mechanisms and techniques are able to detect failures or critical conditions.

Identify Adaptation Need. Some failures or critical conditions become triggers for the SBA to leave “normal” operation and enter the adaptation or evolution cycle. The adaptation cycle is responsible for deciding whether the SBA needs to be adapted in order to maintain its expected functionality and quality (i.e., to meet its requirements). This is an important decision as it might well be that despite a failure of a service, the end-to-end quality of the SBA is not affected and hence there is no need to react to that situation. Such decisions may be made automatically, or it may require human intervention (end user, system integrator, application manager). Moreover, such decisions may be made in a reactive way, when the problem has already occurred, or in a proactive way, where a potential, future problem could be avoided. It should be noted that the decision could also be that there should be an evolution of the system rather than an adaptation, thereby entering the “development and evolution” cycle.

Identify Adaptation Strategy. When the adaptation needs are understood, the corresponding adaptation strategies are identified and selected. Possible types of adaptation strategies include service substitution, SLA re-negotiation, SBA re-configuration or service re-composition. It could also happen that several adaptation strategies are able to satisfy a specific adaptation need. The selection of the strategy and its instantiation (e.g., which service to use as a substitute or which re-configuration to perform) may be automatic if either the SBA or the execution platform decide the action to perform, or it can be done by (the help of) a human operator. Specifically, two questions need to be answered: ”what to adapt?” and ”how to adapt?”.

Enact Adaptation. After the choice of the adaptation strategy, the adaptation mechanisms are used to enact the adaptation. For example, service substitution, re-configuration or re-composition may be obtained using automated service discovery and dynamic binding mechanisms, while re-composition may be achieved using existing automated service composition techniques. Depending on the situation, such an adaptation can be done manually (e.g., by a human operator), semi-automatically or fully-automatically.

III. Application Scenario

In this section, an example workflow is introduced in order to illustrate the problems as well as the solution that will be presented in Section IV. The workflow specifies an eGovernment SBA that allows citizens to pay parking tickets online, thereby saving effort and costs (see [21] for a description of the eGovernment application domain as defined in S-Cube).

A. Workflow

The workflow as well as the service composition of the eGovernment application are depicted in Figure 2 as an extended activity diagram. The gray boxes denote concrete services that can be composed to an eGovernment application. In the example, each service is provided by a third party, being it an external organization or a different unit of the governmental organization. Solid connections between workflow actions and services denote the bindings established at deployment time. Dashed connections denote possible alternative services (from a different provider). In addition, the diagram is annotated with information about the negotiated response times (which could be stipulated by means of SLAs).

Let us assume that the overall workflow is expected to have a response time of at most 1250 ms. This quality requirement can be satisfied by the bound services, provided that they meet their negotiated maximum response times (as, altogether, the maximum response times along the longest path add up to 1200 ms).

In the following subsections we use this example to illustrate the shortcomings of reactive adaptation, which have been introduced in Section I-A. We assume that the *ePay* service of the example workflow fails during runtime, i.e., takes longer than the negotiated maximum response time.

B. Scenario A: Requirements Monitoring

As mentioned in Section I-A there are approaches which are restricted to monitoring of requirements. In that case monitoring events might arrive so late that an adaptation of the SBA is not possible anymore. In our example, the *ePay* service invoked by *Make Payment* might take 650 ms to respond instead of the negotiated maximum response time of 400 ms (see Scenario A in Figure 3).

Due to the fact that only the requirement (maximum response time of 1250 ms) is monitored, this failure is not registered until after *Sign* has been invoked. As a consequence the mechanism was not able to prevent the deviation from the requirements, even though the failure has occurred much earlier (see Δ in Figure 3).

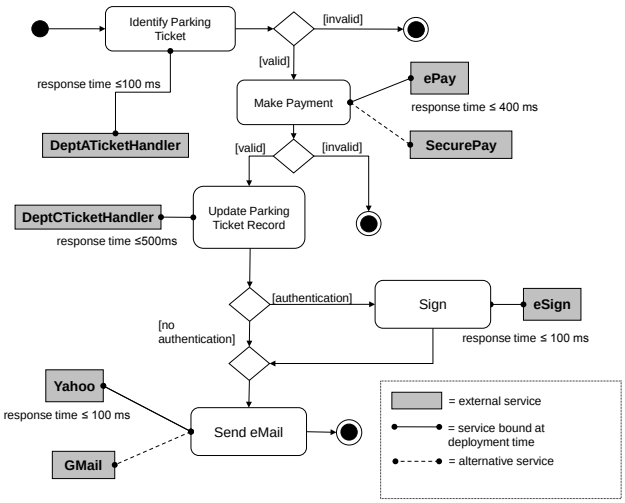


Fig. 2. Workflow of an eGovernment Application

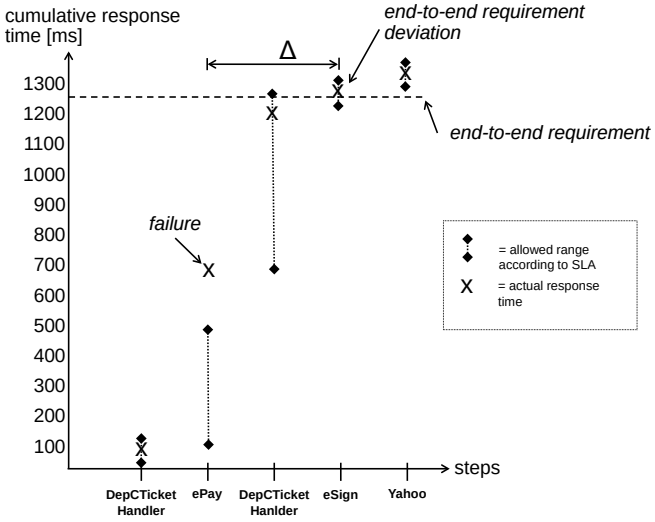


Fig. 3. Scenario A: Requirements Monitoring

C. Scenario B: Service Monitoring

Referring to Section I-A, approaches that monitor individual services exist. However, in such setting it remains unclear whether the failure of a single service leads to a violation of the SBA's requirements. In the example, let us assume that instead of 400 ms the *ePay* service invocation takes 450 ms (see Scenario B in Figure 4).

This failure is observed by means of monitoring and leads to an adaptation of the SBA. However, as obvious in the figure, the overall response time would have still matched the required response time even if no adaptation would have been performed. Thus, in this case an adaptation was triggered although it was not necessary.

In the next section we will present a proactive approach which addresses the above shortcomings.

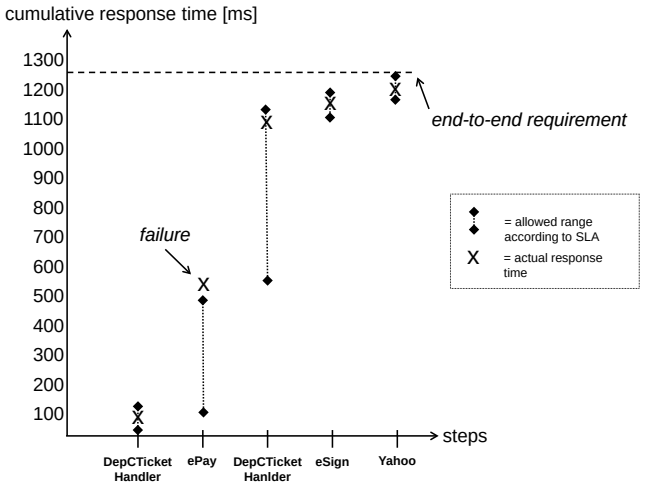


Fig. 4. Scenario B: Service Monitoring

IV. Proactive Adaptation along the Life-Cycle

This section describes techniques developed in S-Cube for enabling proactive adaptation. The description is organized along the phases of the life-cycle model from Section II. In order to illustrate the techniques, we refer to the example SBA and scenarios presented in Section III.

As explained in Section I-A, adaptive SBAs automatically and dynamically adapt to changing conditions and changes of service functionality and quality. To enable such an automatic adaptation, the relevant artifacts, as well as the properties of the SBAs and their context need to be formalized to make them amenable to automated checks and decisions. In the remainder of this section, we thus introduce concrete formalization approaches, as well as techniques that build on this formalization.

A. Requirements Engineering

To automatically assess whether the application deviates from its requirements during operation and thus trigger an adaptation, functional and non-functional requirements need to be collected and formally expressed. We propose to formalize the requirements already in the requirements engineering phase, as this also facilitates an early validation of the requirements, e.g., by means of formal consistency checks (cf. [22]), and hence reduces the risk of expensive corrections in later phases.

S-Cube has developed various approaches to formalize requirements (depending on the actual SBA type). For instance ALBERT is a specification language based on temporal logics (presented in [12]). ALBERT is used to encode functional and quality attributes. In addition, the S-Cube Quality Meta Model (QMM) has been defined, which provides a set of key concepts for expressing quality requirements and constraints (see [23]).

To express the requirements for monitoring, an integrated monitoring framework and the corresponding specification language has been provided (see [24] and [25]) in the scope of S-Cube. The framework integrates the capabilities of two monitoring platforms: Dynamo [11] and ASTRO [9]. On the one hand, the language enables the specification of complex point-wise properties over service composition execution (e.g., pre- and post-conditions on service calls), taking into account current and historical values of the process variables, complex constraints, and event external properties. On the other hand, simple events and point-wise properties may be aggregated into complex behavioral expressions, also taking into account temporal and statistical information necessary for capturing non-functional requirements. While the latter capability is very close to the approach used by ALBERT, the notation allows for expressing properties over classes of processes rather than over single instances. This capability may be very important in order to trigger "evolution" of the workflow, when the problem applies to the whole SBA model rather than to a single SBA instance.

Example: In our example from Section III, we need to formalize the required response time r_{perf} of the eGovernment application. r_{perf} is an element of the given set of requirements R_{eGov} against the eGovernmant application:

$$r_{perf} \in R_{eGov}$$

r_{perf} demands the response time of the eGovernment application to be at most 1250 milliseconds. Due to the capability of ALBERT to express the dependencies of monitoring data along an executed path, we choose this language to specify the requirement (r_{perf}) as follows:

$$\begin{aligned} r_{perf} \quad := \quad & onEvent(start, "Identify\ Parking\ Ticket") \\ & \rightarrow Within(onEvent(end, "Send\ eMail"), \\ & 1250) \end{aligned}$$

The $onEvent$ operator evaluates to true if the activity specified in the second argument performs the state change denoted in the first argument. The $Within$ operator evaluates to true if its first argument evaluates to true within the amount of milliseconds specified in its second argument.

B. Design

Similarly to the requirements, the workflow of the SBA needs to be formalized to support automated checks. Following the same reasoning as in the requirements engineering phase (see above), we suggest to formalize the workflow during the design phase already in order to reduce the risk of later corrections. As presented in [8] the checks can be performed by using Model Checking techniques. In S-Cube the use of BOGOR has been proposed [12] to assess whether the specified SBA satisfies the requirements. We thus formalize the workflow using the input language of the model checker, in this case BIR (Bogor Input Representation).

Example: In order to use the BOGOR Model Checker (as proposed in [12]) we specify the eGovnerment Workflow by using BIR. The resulting specification S_{eGov} (see Listing 1 in the appendix) can be directly executed and analyzed by BOGOR.

C. Realization

During the realization phase, the quality levels (aka. service level objectives) that have been negotiated and agreed upon with the service providers (see Section II), are formalized.

Following the proposal in [8], we treat those quality levels as assumptions (A) about the SBA's context. Due to the lack of control of third-party services, those quality levels could be violated during the operation of the SBA (see Section I). To formalize A, we can use one of the quality formalization approaches as used during the requirements engineering phase.

For checking the violation of the assumptions during the operation of the SBA, monitoring mechanisms are implemented that collect the relevant data (cf. [24], [25] and [12]). This is equivalent to collecting the monitoring data in the reactive case of adaptation (cf. Section III).

Example: According to their SLAs (see Figure 2) ALBERT is used to formalize the five assumed response times. The set of assumptions A_{eGov} for the parking ticket SBA is defined as

$$\begin{aligned} A_{eGov} \quad := \quad & \{a_{DeptATicketHandler}, \\ & a_{ePay}, a_{DeptCTicketHandler}, \\ & a_{eSign}, a_{Yahoo}\} \end{aligned}$$

The assumption a_{ePay}, related to the *ePay* service invocation, is formalized as follows:

$$\begin{aligned} a_{ePay} \quad := \quad & onEvent(start, "Make\ Payment") \\ & \rightarrow Within(onEvent(end, \\ & "Make\ Payment"), 400) \end{aligned}$$

D. Deployment

Before deploying the SBA, it is checked whether the workflow specification (S), under the given assumptions (A), satisfies the requirements (R):

$$S, A \models R$$

This check ensures that the initial composition – the workflow and the services – satisfy the requirements. If this is not the case, the phases of the evolution loop (cf. life-cycle in Section II-A) are executed again in order to redesign the application, e.g., to bind faster services. If the SBA is successfully verified against the requirements the SBA is deployed.

Example: In our example S_{eGov} and A_{eGov} satisfy R_{eGov}. In consequence the SBA is deployed.

E. Operation and Management

This phase comprises the execution and the monitoring of the individual services of the deployed SBA.

Monitoring is supported by monitoring frameworks, such as Dynamo (presented in [25]). During runtime, the monitoring framework continuously assesses whether the monitoring data M satisfies the formalized assumptions A about the services:

$$M \models A$$

If a violation occurs, the SBA enters the adaptation loop (cf. Section II-B). The relevant activities are described below (Sections IV-F, IV-G, and IV-H).

Example: After finishing the SBA deployment, the eGovernment application is executed. Let us assume that the first activity, which invokes the $DeptATicketHandler$ service, lasts 90 milliseconds. The measured response time of the $DeptATicketHandler$ call is stored as monitored data $m_{DeptATicketHandler}$. $m_{DeptATicketHandler}$ satisfies the assumption that the service responds within 100 milliseconds ($a_{DeptATicketHandler}$). In the next step $ePay$ is invoked. Let us assume, that the invocation of $ePay$ is slower than expected. This is the same situation as described in Scenarios A and B (see Section III). Instead of 400 milliseconds as expected, the ePay invocation takes 450 milliseconds (cf. Scenario B). Hence, the monitoring data of the second service invocation m_{ePay} doesn't satisfy the corresponding assumption a_{ePay}:

$$m_{ePay} \not\models a_{ePay}$$

Due to this violation the phases of the adaptation loop are entered.

F. Identify Adaptation Needs

In this phase it is checked, whether the requirements are still satisfied, although the assumptions have been violated (cf. [8]). For example it might be the case that a slower response time of one service is compensated by a faster response time of a previous service, and consequently no adaptation is required.

When the check is performed, there usually are services which have not been invoked. Only when the workflow is finished, all services have been invoked. This means, that there is no monitoring data available for the not yet invoked services. For those not yet invoked services we continue to use their assumptions in the checks, i.e. we use a subset $A' \in A$. Next, it is checked, whether the workflow specification S, the monitored data M and the assumptions in A' satisfy the given requirements R.

$$S, M, A' \models R$$

If R is satisfied, then the workflow execution is continued. If R is not satisfied, the SBA must be adapted.

Example: To illustrate that the presented S-Cube approach is adequate to address the shortcomings from III-B and III-C, we compare the S-Cube approach with the requirements monitoring approach presented in Section III-B (Scenario A) and the sequence monitoring approach presented in Section III-C (Scenario B). It is checked, whether there is a deviation from the requirements, as this could indicate that an adaptation is necessary. This check also covers cases with larger delays, e.g., 500 milliseconds in Scenario A.

The approach presented in Scenario A (see Section III-B) does not realize failures at the moment when they occur – as depicted with Δ in Figure 3. The S-Cube approach does not have this shortcoming. The continuous monitoring of the service behavior registers failures immediately. The SBA

requirements are promptly checked, thus the system can proactively prevent requirement deviations by means of adaptation. In order to eliminate requirement violations Model Checking techniques are used. The workflow specification (S_1), the monitoring data ($m_{DeptATicketHandler}$ and m_{ePay}) together with the assumptions of the outstanding service invocations ($a_{DeptCTicketHandler}$, a_{eSign} and a_{Yahoo}) are checked against the requirement r_{perf}. The expected overall runtime is 1450 milliseconds which exceeds the 1250 milliseconds demanded in r_{perf}. Hence, the requirement r_{perf} is not satisfied. This result is considered as an identified adaptation need. Subsequent to this check, the adaptation can be performed proactively, before the requirement is actually violated.

The approach presented in Scenario B (see Section III-C) is not able to determine, whether a failure of a single service leads to a violation of the SBAs requirements. Every time a service fails, the SBA adapts immediately. The S-Cube approach presented in this paper allows adapting only in cases when critical failures occur, thereby avoiding unnecessary adaptations. The same check as described above assesses that the expected overall runtime does not exceed 1250 milliseconds. The requirement r_{perf} is still satisfied and thus no adaptation trigger is needed. Thereby an unnecessary adaptations is prevented, which would have been performed in Scenario B.

G. Decide on Adaptation / Identify adaptation strategy

When the need for adapting an SBA is detected, the next step is to identify and apply an appropriate adaptation strategy among the ones that are available for the considered applications. Depending on the application, the adaptation strategies may range from service re-execution, over replacement of a single service or of the process fragment, over re-negotiation of quality properties, to changes in underlying infrastructure, etc. Note that the adaptation strategies should be designed with the application since some of them require the adoption of specific infrastructure or the implementation of additional components.

Typically, the adaptation strategy is associated with a specific critical situation or a problem at design time. This association may be done either implicitly or explicitly. In the former case, the mechanisms for choosing one action or another are "hard-coded" in some decision mechanisms. A typical scenario is the replacement of a service that violates the SLA or a SBA requirement with a new one, with appropriate and most suitable characteristics. Based on the selection criteria (e.g., optimization of a quality function, adherence to application constraints), the appropriate decision mechanism may choose one service or another. In the scope of the S-Cube project, several approaches follow this vision. For example, in [26] the replacement policies realize such a decision mechanism and define the association between various types of changes (service failure, changes in service properties and models, appearance of new services, and changes in the context and requirements) and the service selection. In [27], the decision on the adaptation strategy is based on the quality factors of the SBA that should be improved. Those factors are identified through the analysis of the dependency tree that capture the relation between simple quality factors and SBA requirements. At design time, the adaptation action is assigned to the quality factors that it influences either positively or negatively. The selection of the adaptation strategy is based on the need to improve quality factors that are critical for the requirement, while trying to minimize the negative effect on the other factors. In our scenario, the requirement would need to improve the performance of the last service, and the service replacement would be proposed such that the new service has better performance, while having smaller cost with respect to alternatives.

The definition of the adaptation strategy may be also explicitly assigned to the critical situation. For example in [28] the adaptation strategy is represented in the WS-ReL, a notation for specifying and integrating recovery actions in service composition. Therefore, the adaptation is defined as a rule, where in the left hand side a critical situation is defined (as a formal requirement to be monitored) and in the right hand side a set of actions to be applied. The possible actions include re-execution of a service invocation, replacement of a service or a provider (partner link), ignoring the failure or halting the execution, executing an extra process fragment, or rolling back to a safe point. Simple actions may be joined into a complex strategy by defining a control flow over actions, like "try action A else try action B and action C". These rules are evaluated and applied by the underlying adaptation engine.

Adaptation can also be based on the causes of failures. This is particularly helpful when invoked services are stateful, and their invocation modifies the state of the service, such as in transactional services. For processes involving transactional services, if a diagnosis mechanism is available, such as in [29], the adaptation strategy can depend on the cause of the failure and its implications on the processes. This might imply an adaptation strategy involving one or more services in the process which must be dynamically generated.

H. Enact Adaptation

To enact adaptation actions, the SBA or its execution platform should be appropriately instrumented. A typical approach for realizing adaptation mechanisms for SBAs implemented as executable (BPEL) processes is to instrument the process execution engine. Such instrumentation is done via Aspect-Oriented Programming techniques, as the adaptation activities are treated as a cross-cutting concern. Using this approach, the join points allow for injecting the adaptation logic in order to intercept and adjust process execution logic. In particular, in [28] a supervision manager component is attached to the ActiveBPEL process engine and performs the necessary supervision activities: monitoring of critical situations, evaluation of adaptation rules, and calls to the process engine infrastructure to realize the specific strategy. Similarly, in [30], where aspect-oriented techniques are adopted in order to dynamically bind services into service compositions that are realized as BPEL orchestrations.

V. Conclusion and Perspectives

This paper has introduced novel techniques developed in S-Cube (the European Network of Excellence on Software, Services and Systems) for equipping service-based applications with proactive adaptation facilities. Thereby, those techniques are able to avoid costly compensation and repair activities, as well as unnecessary adaptations, which are deemed key shortcomings of current solutions for adaptive service-oriented applications.

The techniques have been introduced along the key phases of the S-Cube service life-cycle. Thereby, this paper has demonstrated when and how those techniques should be applied when developing, evolving and adapting service-based applications.

We are confident that those techniques will become especially relevant in the setting of the "Internet of Services", where applications will increasingly be composed from third party services, which are not under the control of the service consumer. This implies that applications and their constituent services need to be continuously checked during their operation such that they can be dynamically adapted or evolved in order to respond to failures or unexpected changes of third party services.

In S-Cube, we are currently striving to push the envelope towards proactive adaptation even further. In addition to determining the need for adapting the service-based application based on actual failures of the application's constituent services, we investigate the applicability of online testing for predicting the quality of those services (e.g., see [6], [7]). Combined with the approaches introduced in this paper, this means that critical problems could be observed even earlier, thus enabling a broader range of adaptation and evolution strategies. For instance, in our running example we can only react to the violation of the response time of a constituent service by ensuring that the remainder of the workflow executes faster. However, if the quality prediction techniques forecast a violation of the expected response time of a specific service, this very service can be replaced before it is invoked in the context of the service-based application.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube). For further information please visit http://www.s-cube-network.eu/.

References

[1] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl, "A journey to highly dynamic, self-adaptive service-based applications," *Automated Software Engineering*, 2008.

[2] G. Canfora and M. Di Penta, "Testing services and service-centric systems: Challenges and opportunities," *IT professional*, vol. 8, no. 2, pp. 10–17, 2006.

[3] A. Metzger and K. Pohl, "Towards the next generation of service-based systems: The s-cube research framework," in *CAiSE 2009*, ser. LNCS, J. P. van Eck, J. Gordijin, and R. Wieringa, Eds. Berlin Heidelberg: Springer-Verlag, 2009, pp. 11–16.

[4] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 2, 2009.

[5] S. Benbernou, "State of the art report, gap analysis of knowledge on principles, techniques and methodologies for monitoring and adaptation of sbas," S-Cube Consortium, Deliverable PO-JRA-1.2.1, July 2008. [Online]. Available: http://www.s-cube-network.eu/results/

[6] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka, "Towards proactive adaptation with confidence augmenting service monitoring with online testing," in *Proceedings of the ICSE 2010 Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS '10)*, Cape Town, South Africa, 2-8 May 2010.

[7] J. Hielscher, R. Kazhamiakin, A. Metzger, and M. Pistore, "A framework for proactive self-adaptation of service-based applications based on online testing," in *ServiceWave 2008*, ser. LNCS, no. 5377. Springer, 10-13 December 2008.

[8] A. Gehlert, A. Bucchiarone, R. Kazhamiakin, A. Metzger, M. Pistore, and K. Pohl, "Exploiting assumption-based verification for the adaptation of service-based applications," in *SAC '10: Proceedings of the 2010 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2010, pp. 2430–2437.

[9] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-Time Monitoring of Instances and Classes of Web Service Compositions," in *IEEE International Conference on Web Services (ICWS 2006)*, 2006, pp. 63–71.

[10] D. Dranidis, E. Ramollari, and D. Kourtesis, "Run-time verification of behavioural conformance for conversational web services," in *Seventh IEEE European Conference on Web Services (ECOWS 2009), 9-11 November 2009, Eindhoven, The Netherlands*, R. Eshuis, P. W. P. J. Grefen, and G. A. Papadopoulos, Eds., 2009, pp. 139–147.

[11] C. Ghezzi and S. Guinea, "Run-time monitoring in service-oriented architectures," in *Test and Analysis of Web Services*, L. Baresi and E. D. Nitto, Eds., 2007, pp. 237–264.

[12] D. Bianculli, C. Ghezzi, P. Spoletini, L. Baresi, and S. Guinea, *Advances in Software Engineering*, ser. Lecture Notes in Computer Science. Springer-Verlag, 2008, vol. 5316, ch. A Guided Tour through SAVVY-WS: a Methodology for Specifying and Validating Web Service Compositions, pp. 131–160.

[13] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime prediction of service level agreement violations for composite services," in *3rd Workshop on Non-Functional Properties and SLA Management in Service-Oriented Computing, co- located with ICSOC 2009*, 2009.

[14] A. Bucchiarone, C. Cappiello, E. D. Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore, "Design for adaptation of service-based applications: Main issues and requirements," in *Fifth International Workshop on Engineering Service-Oriented Applications: Supporting Software Service Development Lifecycles (WESOA)*, 2009.

[15] A. Gehlert, M. Pistore, P. Plebani, and L. Versienti, "First version of integration framework," S-Cube Consortium, Deliverable CD-IA-3.1.3, December 2009. [Online]. Available: http://www.s-cube-network.eu/results/

[16] A. Bucchiarone, R. Kazhamiakin, C. Cappiello, E. Di Nitto, and V. Mazza, "A context-driven adaptation process for service-based applications," in *PESOS 2010 - 2nd International Workshop on Principles of Engineering Service-Oriented Systems. (To Appear)*, Cape Town, South Africa, 1-2 May 2010.

[17] B. Pernici, *Methodologies for Design of Service-Based Systems*. Springer, 2010, ch. 17.

[18] E. Nitto, "State of the art report on software engineering design knowledge and survey of hci and contextual knowledge," Deliverable PO-JRA-1.1.1, 2008. [Online]. Available: http://www.s-cube-network.eu/results/

[19] D. Bianculli, C. Ghezzi, and C. Pautasso, "Embedding continuous lifelong verification in service life cycles," in *Proceedings of Principles of Engineering Service Oriented Systems (PESOS 2009), co-located with ICSE 2009, Vancouver, Canada*. IEEE Computer Society Press, May 2009.

[20] M. Comuzzi and B. Pernici, "A framework for qos-based web service contracting," *ACM Transactions on web*, vol. 3, no. 3, 2009.

[21] E. D. Nitto, V. Mazza, and A. Mocci, "Collection of industrial best practices, scenarios and business cases," S-Cube Consortium, Deliverable CD-IA-2.2.2, 2009. [Online]. Available: http://www.s-cube-network.eu/results/

[22] F. L. Bauer, R. Berghammer, M. Broy, W. Dosch, F. Geiselbrechtinger, R. Gnatz, E. Hangel, W. Hesse, B. Krieg-Brückner, A. Laut, T. Matzner, B. Möller, F. Nickl, H. Partsch, P. Pepper, K. Samelson, M. Wirsing, and H. Wössner, *The Munich Project CIP: Volume I: the wide spectrum language CIP-L*. London, UK: Springer-Verlag, 1985.

[23] A. Gehlert and A. Metzger, "Quality reference model for SBA," Deliverable CD-JRA-1.3.2, 2008. [Online]. Available: http://www.s-cube-network.eu/results/

[24] L. Baresi, S. Guinea, R. Kazhamiakin, and M. Pistore, "An integrated approach for the run-time monitoring of bpel orchestrations," in *ServiceWave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 1–12.

[25] L. Baresi, S. Guinea, M. Pistore, and M. Trainotti, "Dynamo + astro: An integrated approach for bpel monitoring," *Web Services, IEEE International Conference on*, vol. 0, pp. 230–237, 2009.

[26] K. Mahbub and A. Zisman, "Replacement Policies for Service-Based Systems," in *2nd Workshop on Monitoring, Adaptation and Beyond (MONA+), co- located with ICSOC 2009*, 2009.

[27] R. Kazhamiakin, B. Wetzstein, D. Karastoyanova, M. Pistore, and F. Leymann, "Adaptation of service-based applications based on process quality factor analysis," in *Proc. Of 2nd Intl. workshop on Monitoring, Adaptation and Beyond (MONA+)*, Collocated with ICSOC/ServiceWave'09, 2009.

[28] L. Baresi, S. Guinea, and L. Pasquale, "Integrated and composable supervision of bpel processes," in *International Conference on Service-Oriented Computing (ICSOC)*, 2008, pp. 614–619.

[29] G. Friedrich, M. Fugini, E. Mussi, B. Pernici, and G. Tagni, "Exception handling for repair in service-based processes," *IEEE Trans. Software Eng.*, vol. 36, no. 2, pp. 198–215, 2010.

[30] D. Karastoyanova and F. Leymann, "Bpel'n'aspects: Adapting service orchestration logic," in *International Conference on Web Services (ICWS)*, 2009, pp. 222–229.

APPENDIX

Listing 1. Workflow Specification S_{eGov}

```
system ParkingTicketSpec {
 Action identifyParkingTicket;
 Action makePayment;
 Action updateParkingTicketRecord;
 Action sign;
 Action sendEMail;

 int sumMaxResponseTime := 0;

 record Action {
  string serviceName;
  int maxResponseTime;
  boolean serviceInvoked;
 }

 active thread MAIN () {
  init();
  checkWorkflow();
  checkRequirements();
 }

 function init() {
  identifyParkingTicket :=
  createAction("DeptATicketHandler", 100);
  makePayment := createAction("ePay", 400);
  updateParkingTicketRecord :=
   createAction("DeptCTicketHandler", 500);
  sign := createAction("eSign", 100);
  sendEMail := createAction("Yahoo", 100);
 }

 function checkWorkflow() {
  executeAction(identifyParkingTicket);
  choose
   do skip;
   do
    atomic
     executeAction(makePayment);
     choose
      do skip;
      do
       atomic
        executeAction(updateParkingTicketRecord);
        choose
         do executeAction(sign);
         do skip;
        end
        executeAction(sendEMail);
       end
     end
    end
   end
  }

 function checkRequirements() {
  assert sumMaxResponseTime <= 1250;
 }

 function executeAction(Action action) {
  sumMaxResponseTime :=
        sumMaxResponseTime +
        action.maxResponseTime;
  action.serviceInvoked := true;
 }

 function createAction(
  string serviceName,
  int maxResponseTime) returns Action{
  Action action;
  action := new Action;
  action.serviceName := serviceName;
  action.maxResponseTime := maxResponseTime;
  action.serviceInvoked := false;
  return action;
 }
}
```

A Pattern-based Approach for Monitor Adaptation

Ricardo Contreras
School of Informatics
City University London
London EC1V 0HB, UK
Ricardo.Contreras.1@soi.city.ac.uk

Andrea Zisman
School of Informatics
City University London
London EC1V 0HB, UK
a.zisman@soi.city.ac.uk

Abstract

Monitoring of service-based systems can be used to assist with the verification of the behaviour of the system, and the quality and contextual information of the services participating in a system. In this paper, we present a pattern-based approach to support monitor adaptation: adaptation of monitor rules used by a monitor tool. Our work focuses on monitor adaptation due to changes in the context characteristics of the user interacting with the system and the set of services used by the system. The monitor adaptation is based on the use of patterns represented in Event Calculus for different user context types such as role, skills, needs, preferences, and cognition, as well as location, time, and environment.

1. Introduction

Monitoring of service-based systems has been recognized as a very important activity to support service-oriented computing. More specifically, monitoring of service-based systems is concerned with the activity of collecting information about the execution of a service-based system and verifying if the system is operating correctly by comparing the collected information with the properties of the system (viz. monitor properties or monitor rules).

The monitor can be used to verify the behaviour of a service-based system [1][2][5][32][25], the quality of service aspects of the services participating in a system [11][18][19][20][28], and the contextual information of the services participating in the system and the system itself [4][6][8]. Monitoring of service-based systems is used to support several activities such as (a) adaptation, repair, and evolution of service-based systems; (b) discovery and replacement of services participating in the system; (c) notification of violations of service level agreements, business rules and KPI values; or (d) optimization of resource allocations.

Several approaches have been proposed to support monitoring of service-based systems. These approaches can be classified in terms of the type of information being monitored (*what*), the purpose of the monitoring activity (*why*), and the techniques used to support the monitoring (*how)* [3]. Existing monitoring techniques can be classified as intrusive [2] or non-intrusive [1][23][25][32]. Intrusive techniques are characterized by the use of instrumentation of the service-based system or its constituent services, while non-intrusive techniques are characterized by the use of special components to capture runtime information about the services and the utilization of this information by an external monitor component.

All the existing approaches for monitoring service-based systems assume that the monitor properties (monitor rules) are pre-defined and known in advance. However, this is not always the case given that during execution time of a service-based system it is possible to have changes in the (i) set of services used by the system, (ii) types of interaction of the users with the system, and (iii) context characteristics of the user interacting with the system.

As an example, consider a *financial planner* service-based system that provides financial information including information about stock markets, allows purchasing and selling of shares in different companies, and provides instant messaging services among the users of the planner system. An experienced (skills) broker (role) uses this application to receive constant information about mining companies in Asia. Assume that after a while, the user of the application not only retrieves information from different companies in Asia, but also starts purchasing shares from these different companies (case (ii) above). In this case, new monitor rules relevant to the purchase of shares, including access to the user's bank account, are required (needs). Assume that the broker has used this service-based application in the past for companies in South America. Consider also that historical data about the activities that this broker has executed using this application, when dealing with companies in South America, shows that when the values of the stocks purchased by the user reaches a certain threshold, the user starts to sell the stocks (user's needs). In this case, the historical data together with the fact that the user started to purchase shares in companies are used to forecast possible future activities of the user and, in some situations, require the monitor to be able to give support for the new activity (i.e., selling of shares) (case (iii) above). Moreover, suppose the situation that after a while, the service in the system that provides information about stock markets becomes unavailable. Assume that a new service that provides information about stock markets, as well as analyses the companies and predicts possible outcomes for the shares of the companies is used to replace the initial service. In this case, the monitor system needs to verify the new functionality (i.e., analysis and predictions) (case (i) above).

Therefore, it is necessary to provide ways to support *monitor adaptation*. We define monitor adaptation as adaptation of the monitor rules used by the monitor system. The adaptation of the monitor rules can be executed in three different ways, namely (a) by dynamic selection of the monitor rules to be used, (b) by automatic or semi-automatic modifications of monitor rules, and (c)

by automatic or semi-automatic creation of new monitor rules, triggered by any of cases (i) to (iii) above.

In this paper we present a pattern-based approach to support monitor adaptation as defined above. More specifically, our approach is concerned with HCI-aware monitor adaptation in which changes in the monitor rules are based on user's interaction with a service-based system and different types of user context. The user context information includes role, skills, needs, preferences, and cognitive information of the user represented in user models based on an ontology that we have created. We have also created patterns for monitor rules representing the different context types.

The remainder of this paper is structured as follows. Section 2 describes an overview of our approach, including our user context ontology and user models. Section 3 presents the different context-aware patterns and examples of monitoring rules. Section 4 presents existing related work. Finally, Section 5 summarizes the work and describes material for future work.

2. Overview of Our Approach

Figure 1 shows an overview of the process of our monitor adaptation approach. As shown in the figure, the characteristics of the users are represented in special models. These models together with (i) patterns representing the various types of context related rules and (ii) existing monitor rules are used by the monitor adaptor to provide new monitor rules. More specifically, the monitor adaptor uses the models and the patterns to identify monitor rules[1], to change existing monitor rules, or to create new monitor rules. The new set of monitor rules is used by the monitor to support verification of the service-based system. We assume the use of the monitoring tool described in [32]. However, our approach can be used with any monitoring tool that makes use of monitor rules represented in Event Calculus.

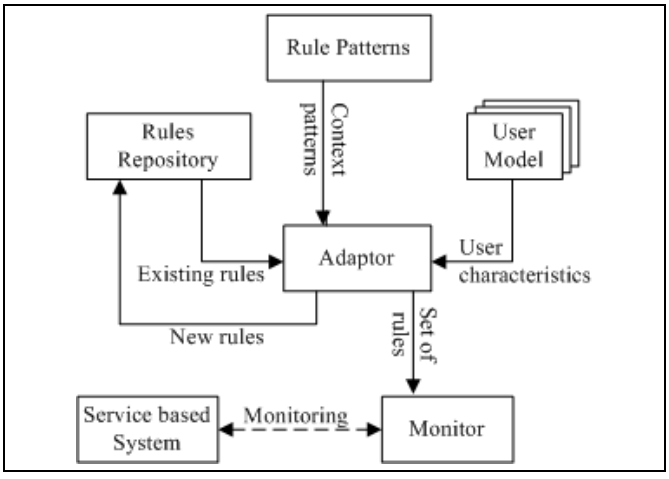


Figure 1: Overview of the approach

The user models are described in an XML format, and are based on the ontology we have created to specify user context types. We have built the ontology using protégé [26] (see Subsection 2.1).

The monitor rules are described in Event-Calculus (EC) [30]: a language based on first-order logic, which can be used to support behaviour representation of dynamic systems. The use of Event Calculus (EC) to describe monitor rules has been advocated in [11][20][32] and has shown to be appropriate to support the representation of several types of rules. More specifically, Event Calculus allows (i) rules to be represented as first order logic, which provides sufficient expressiveness for a large range of applications; (ii) specification of quantitative temporal constraints and relationships that are necessary to be taken into consideration when monitoring service-based systems; (iii) distinction between events and states that are necessary to describe the behavior of a system and interaction of users with the system; and (iv) definition of the influences between events and states despite the possibility of using multiple states and events.

As mentioned before, our approach supports the adaptation of monitor rules in three different ways, ranging from the (a) dynamic selection of monitor rules, to (b) automatic or semi-automatic modifications of existing monitor rules, and (c) automatic or semi-automatic creation of new monitor rules. Any of cases (a)-(c) above can be triggered by changes in user context and changes in the services used in a service-based system. The adaptation of the monitor rules is executed by trying option (a), followed by option (b), and finally by option (c), when a rule does not exist or cannot be modified.

We describe below the ontology we have proposed for the representation of the different types of user context, and the model that we use to represent user information.

2.1 Ontology for User Context

User context is concerned with information about users that interact with service-based systems. Based on the classification proposed in [10][14][29], we classify information about users in two groups, namely (i) *direct* user context types and (ii) *related* user context types. The direct user context types represent information of the characteristics of the users and include *role*, *skills*, *needs*, *preferences*, and *cognitive* context types. The related user context types represent information that may influence user information and include *time*, *location*, and *environment* context types. We define below these various types of direct and related user context.

Role: It signifies a social behaviour of an individual within the domain of a service-based system. The roles of an individual can be concerned with the accessibility to the system, occupation of the user, privileges that the user may have to the system.

Skills: It signifies the level of expertise of an individual with respect to a service-based system. The skills of a user are directly related to the user knowledge and experience with the system. The skills can be defined in terms of the level of expertise of the user (e.g., beginner, average, advanced) or the years of experience.

[1] In our approach the identified, modified and created rules are stored in a rule repository

Preferences: It signifies an individual's choice over pre-established alternatives of computational resources, of a service-based system. Examples of these preferences are concerned with security, reliability, response time, availability, and cost characteristics of a service-based system.

Needs: It signifies what an individual wants or requires from a service-based system.

Cognition: It signifies individual's characteristics associated to the process of thought. It is concerned with the way that individuals think, feel, or react. Examples of these characteristics are reaction, user attention level, and user comprehensive ability.

Time: It signifies all possible types of information related to the moment when the user interacts with a service-based system such as hour, date, day, week, or season.

Location: It signifies information related to the place where the user interacts with a service-based system such as coordinates, city, and country.

Environment: It signifies information concerned with the environment where the system is being used. This context includes information such as temperature, traffic conditions, or climate.

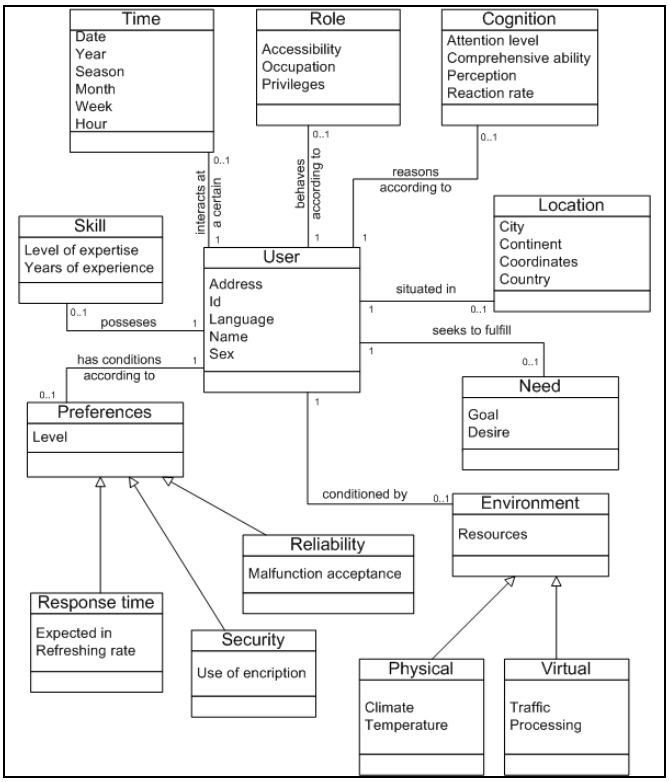


Figure 2: User Context Ontology

Figure 2 shows a general graphical representation of our user context ontology. As shown in Figure 2, the different context types are represented as classes, with subclasses in some cases, and are associated with a central class representing the user. These associations represent relationships between the different attributes of a context type (e.g., occupation for context class role) and a user class. For each class their attributes and respective data types are presented inside the class. Please note that at this stage we do not consider associations between the different context types given that the focus of the work is on relationships between users and contexts and not between context and context.

The user class represents information about the user ranging from unique identification (e.g., user ID, user name) to profile information (e.g., sex, language, address), and the associations between a user class and the other context type classes.

Some of the attributes in the ontology are defined as *symbols* of values (e.g., low, medium, high; male, female; slow, normal, fast; beginner, average, expert), while other attributes support definition of specific values represented as string, integer, or real data types.

2.2 User Models

Our user models are represented in XML format. An example of a user model based on the ontology shown in Figure 2 is presented in Figure 3.

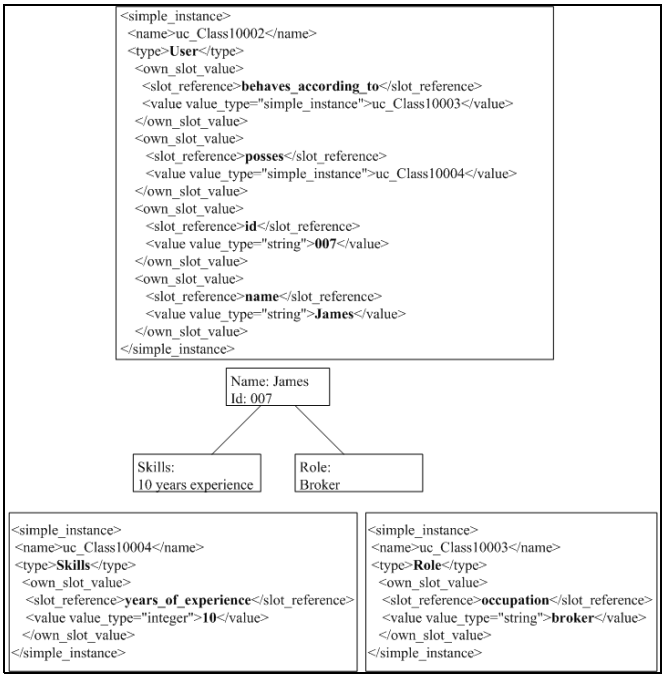


Figure 3: Example of a User Model

The example in Figure 3 is based on the scenario described in Section 1. As shown in the figure, a user called James, with unique identifier 007, is a broker (role) with 10 years of experience (skills). In this example different attributes from the *skill* and *role* context types and some *user* attributes are specified in the model. The attributes specified for skill and role context types are associated with the user through relations *possesses* and *behaves according to,* as shown in the figure.

3. Context-aware Monitoring Patterns

3.1 Overview of Event Calculus

As discussed in Section 2, our work assumes monitor rules described in Event Calculus (EC) [30]. Due to space

limitations we restrict the description of EC elements to those that are used in the patterns we are proposing.

EC makes use of *events* and *fluents* to represent the behaviour of a system. An event occurs at a specific instance of time and may change the state of a system. A fluent is a condition of a system state and may be affected by the occurrences of events. Both events and fluents are represented in EC by predicates.

The occurrence of an *event*, at some time *t*, is represented by the predicate *Happens(event,t,R(t1,t2))*, which means an *event* occurs at a time *t*, where *t* is within an interval between *t1* and *t2*. The time boundaries, represented by *t1* and *t2*, can be specified using time variables or arithmetic expressions over time variables, and represent the lower and upper time boundaries.

The initialization of a *fluent*, is represented by the predicate *Initiates(event,fluent,t)*, which means a *fluent* starts to holds after an *event* occurs at a time *t*. The predicate *HoldsAt(fluent,t)*, means a *fluent* holds (is valid) at a time *t*. The termination of a *fluent*, is represented by the predicate *Terminates(event,fluent,t)*, which means a *fluent* ceases to hold after an *event* occurs at a time *t*.

We have also classified the different types of fluents and events as: *user-triggered-event*, representing events directly triggered by the user; *system-triggered-event*, representing events that are not directly triggered by the user; *an-event*, representing any event; *fluent*, representing any fluent and *user-dependent-fluent*, representing a fluent that holds (or not) depending on the occurrence of a user-triggered-event.

3.2 Patterns

We have developed patterns of monitor rules for different user contexts named *context patterns*. Our context patterns are composed of two parts, namely (a) *monitor rules* and (b) *assumptions*. The monitor rules represent properties of a service-based system that need to be monitored, while the assumptions represent event calculus formulae that need to be used to identify state information of the system. The state information will be used by the monitor rules. In the following we describe different patterns for the different types of user context of our concern.

3.2.1 Pattern for Role

The pattern for role context type has been proposed based on the belief that, depending on the role of a user, the same user action can activate different processes in a system. In this case, it is necessary to have two or more monitor rules concerned with the same user event (user-triggered-event) implying different system events (system-triggered-event). For example, in the financial planner service-based system described in Section 1, a user that is a broker (occupation in role) wants to purchase shares from a mining company in Asia, while another user that is a finance teacher (occupation in role) wants to collect information about mining companies in Asia to prepare for a lecture.

Figure 4 presents the pattern for role context type. The monitor rule part in this pattern states that after the same user event is triggered (user-triggered-event-A) a different system event should be triggered depending on the role (system-triggered-event-A and system-triggered-event-B).

The assumption part in this pattern states that when different system events occur different fluents are initiated.

monitoring rule: **Happens** (user-triggered-event-A, t1, R(t1,t1)) → **Happens** (system-triggered-event-A, t3, R(t1,t2)) *assumption:* **Happens** (system-triggered-event-A, t1, R(t1,t1)) → **Initiates** (system-triggered-event-A, fluent-A, t1)
monitoring rule: **Happens** (user-triggered-event-A, t1, R(t1,t1)) → **Happens** (system-triggered-event-B, t3, R(t1,t2)) *assumption:* **Happens** (system-triggered-event-B, t1, R(t1,t1)) → **Initiates** (system-triggered-event-B, fluent-B, t1)

Figure 4: Pattern for role context type

Example: To illustrate the use of the role context type pattern, consider the scenario of the service-based system described in Section 1 and the broker and the finance teacher above. Figure 5 shows examples of monitoring rules related to the two roles above for the role patterns shown in Figure 4. According to the rules in Figure 5, the same user action *get mining company* can trigger the retrieval of company information or a connection to a bank to support the purchase of shares if the role of the user is a teacher or broker respectively.

monitor rule: **Happens** (get-mining-company, t1, R(t1,t1)) → **Happens** (retrieve-comp-information, t2, R(t1,t1+100)) *assumption:* **Happens** (retrieve-comp-information, t1, R(t1,t1)) → **Initiates** (retrieve-comp-information, reports, t1)
monitor rule: **Happens** (get-mining-company, t1, R(t1,t1)) → **Happens** (connect-bank-account, t2, R(t1,t1+100)) *assumption:* **Happens** (connect-bank-account, t1, R(t1,t1)) → **Initiates** (connect-bank-account, secure-connection, t1)

Figure 5: Example of monitoring rules for role context type

3.2.2 Pattern for Skill

The pattern for skill context type has been proposed based on the belief that different user actions, capable of activating the same process in a system, occur depending on the skill of the user. In this case, it is necessary to have two or more monitor rules concerned with different user events implying the same system event. For example, in the financial planner system, an expert user (experience level in skills) could retrieve information specifying fields of interest, while a beginner would use an assistant for specifying the same fields of interest.

monitor rule:
Happens (user-triggered-event-A, t1, R(t1,t1)) →
Happens (system-triggered-event-A, t3, R(t1,t2))
assumption:
Happens (system-triggered-event-A, t1, R(t1,t1)) →
Initiates (system-triggered-event-A, fluent-X, t1)

monitor rule:
Happens (user-triggered-event-B, t1, R(t1,t1)) →
Happens (system-triggered-event-A, t3, R(t1,t2))
assumption:
Happens (system-triggered-event-A, t1, R(t1,t1)) →
Initiates (system-triggered-event-A, fluent-X, t1)

Figure 6: Pattern for skill context type

Figure 6 presents the pattern for skill context type. The monitor rule part in this pattern states that after different user events are triggered, the same system event should be triggered. The assumption part in this pattern states that when a system event occurs it initiates a specific fluent.

Example: To illustrate the use of the skill context type pattern, consider the planner scenario. A skillful user would retrieve information from mining companies using the specific feature provided for this purpose. An unskillful user would retrieve the same information using a guided assistant. Figure 7 shows examples of monitor rules related to the skill patterns shown in Figure 6. According to the rules in Figure 7 different user actions, manual-selection and guided-selection can trigger the retrieval of the same information.

monitor rule:
Happens (manual-selection, t1, R(t1,t1)) →
Happens (retrieve-info, t2, R(t1,t1+100))
assumption:
Happens (retrieve-info, t1, R(t1,t1)) →
Initiates (retrieve-info, deliver-companies-info, t1)

monitor rule:
Happens (guided-selection, t1, R(t1,t1)) →
Happens (retrieve- info, t2, R(t1,t1+100))
assumption:
Happens (retrieve- info, t1, R(t1,t1)) →
Initiates (retrieve-info, deliver-companies-info, t1)

Figure 7: Example of monitoring rules for skill context type

3.2.3 Pattern for Preference

For the preference context type we focused in particular on the *response time*. This context type represents the user's time flexibility when waiting for the system response. The pattern has been proposed on the beliefs that a user action can trigger different processes in the system; however the activation of a particular process is essential for other processes to occur. It is also assumed that the essential process is performed quicker than the other processes.

In this case, it is necessary to have two or more monitor rules concerned with the same user event implying different system events; and those system events triggering the same state. For example, in the planner scenario, a user retrieves stock information in a low resolution basic text form if the response time is a concern (level high in response time), or in a high resolution graphics, if time is not a concern (level low in response time). In this example, high resolution depends on low resolution.

Figure 8 presents an example of a pattern for response time preference context type. The monitor rule part in this pattern states that after the same user event is triggered, different system events are triggered.

The assumption part in this pattern states the initialization of a fluent triggered by the different system events and system events dependency.

monitor rule:
Happens (user-triggered-event-A, t1, R(t1,t1)) →
Happens (system-triggered-event-A, t3, R(t1,t2))
assumption:
Happens (system-triggered-event-A, t1, R(t1,t1)) →
Initiates (system-triggered-event-A, fluent-A, t1)

monitor rule:
Happens (user-triggered-event-A, t1, R(t1,t1)) →
Happens (system-triggered-event-B, t3, R(t1,t2))
assumption:
Happens (system-triggered-event-B, t1, R(t1,t1)) →
Happens (system-triggered-event-A, t3, R(t1,t2)) ∧
Initiates (system-triggered-event-B, fluent-A, t3)

Figure 8: Pattern for time preference context type

Example: To illustrate the use of the time preference context type pattern, consider the planner scenario and a user that would like to receive updated information about company shares as fast as possible and a user that would like to receive the same information in a detailed graphical format where time is not a concern. Figure 9 shows examples of monitor rules related to the response time patterns shown in Figure 8.

According to the rules in Figure 9, the same user action, request updates, can trigger different actions associated to the way the information is displayed (i.e. low and high quality display). The high quality however is dependent on the low quality and, because of this dependency; it is assumed it takes more time than a low quality display.

monitor rule:
Happens (request-updates, t1, R(t1,t1)) →
Happens (low-quality-display, t2, R(t1,t1+100))
assumption:
Happens (low-quality-display, t1, R(t1,t1)) →
Initiates (low-quality-display, show-acc-to-quality, t1)

monitor rule:
Happens (request-updates, t1, R(t1,t1)) →
Happens (high-quality-display, t2, R(t1,t1+100))
assumption:
Happens (high-quality-display, t1, R(t1,t1)) →
Happens (low-quality-display, t3, R(t1,t2)) ∧
Initiates (high-quality-display, show-acc-to-quality, t3)

Figure 9: Example of monitoring rules for time preference context type

3.2.4 Pattern for Cognition

The pattern for the cognitive context type has been proposed based on the belief that according to the user

cognition, a user interaction takes more or less time to occur, affecting a process requiring user intervention. In this case it is necessary to have a monitor rule concerned with the occurrence of a user event in a defined period of time.

Figure 10 presents the pattern for the cognitive context type. The monitor rule part in this pattern states that when a fluent is dependent on a user event, the user event should occur in a defined period of time. The assumption part in this pattern states that the user dependent fluent can be initiated by any event and if no user event occurs in a defined time, the user dependent fluent ends.

monitor rule:
HoldsAt (user-dependent-fluent, t1) →
Happens (user-triggered-event, t3, R(t1,t2))
assumptions:
Happens (an-event-A, t1, R(t1,t1)) →
Initiates (an-event-A, user-dependent-fluent, t1)

HoldsAt (user-dependent-fluent, t1) ∧
¬**Happens** (user-triggered-event, t3, R(t1,t2)) →
Happens (system-triggered-event-A, t2, R(t2,t2))

Happens (system-triggered-event-A, t1, R(t1,t1)) →
Terminates (system-triggered-event-A, user-dependent-fluent, t1, R(t1,t1))

Figure 10: Pattern for cognitive context type

Example: To illustrate the use of the cognitive context type pattern, consider the financial planner scenario and a user that checks a bank transaction before authorizing the funds transfer. Assume the user has been checking transfers for the last 10 hours and her productivity decreases (decrement in the user reaction rate).

Figure 11 shows an example of monitor rule related to the cognitive pattern shown in Figure 10. According to the monitor rule in Figure 11, the user transaction confirmation should occur in the specified time or the bank account ceases to be active.

monitor rule:
HoldsAt (bank-account-active, t1) →
Happens (user-trans-confirmation, t2, R(t1,t1+10))
assumptions:
Happens (stock-purchase, t1, R(t1,t1)) →
Initiates (stock-purchase, bank-account-active, t1)

Holds (bank-account-active, t1) ∧
¬**Happens** (user-trans-confirmation, t2, R(t1,t1+10))→
Happens(interact-time-exceeded, t3, R(t1+10,t1+10))

Happens (interact-time-exceeded, t1, R(t1,t1)) →
Terminates (interact-time-exceeded, bank-account-active, t1, R(t1,t1))

Figure 11: Example of monitoring rules for cognitive context type

3.2.5 Pattern for Needs

The pattern for need context type has been proposed based on the belief that a user is able to choose among actions in a system. Since the need concept is too general to be represented, we have limited our identification of *needs* to mutually exclusive system actions (e.g. use or no use of graphical interface). For this we focus on the identification of two rules that may be incompatible.

Figure 12 presents the pattern for the need context type. The monitor rule part in this pattern states that after the same user event is triggered the same system event should either occur or not occur. The assumption part in this pattern states that the occurrence of a system event initiates a fluent.

monitor rule:
Happens (user-triggered-event, t1, R(t1,t1)) →
Happens (system-triggered-event, t3, R(t1,t2))
assumptions:
Happens (system-triggered-event, t1, R(t1,t1)) →
Initiates (system-triggered-event, fluent-A, t1)

monitor rule:
Happens (user-triggered-event, t1, R(t1,t1)) →
¬**Happens** (system-triggered-event, t3, R(t1,t2))
assumptions:
Happens (system-triggered-event, t1, R(t1,t1)) →
Initiates (system-triggered-event, fluent-A, t1)

Figure 12: Pattern for needs context type

Consider, in the planner scenario, that the messaging feature is automatically disabled when performing a transaction and enabled after the transaction has finished.

monitor rule:
Happens (confirm-and-logout, t1, R(t1,t1)) →
Happens (restart-messaging, t2, R(t1,t1+100)):
assumptions:
Happens (restart-messaging, t1, R(t1,t1)) →
Initiates (restart-messaging, messaging, t1)

monitor rule:
Happens (confirm-and-logout, t1, R(t1,t1)) →
¬**Happens** (restart-messaging, t2, R(t1,t1+100))
assumptions:
Happens (restart-messaging, t1, R(t1,t1)) →
Initiates (restart-messaging, messaging, t1)

Figure 13: Example of monitoring rules for need context type

Example: To illustrate the use of the needs context type pattern, consider a user who wants to purchase shares based on information received from colleagues in the form of messages. Assume that after a while the user realizes the information from her colleagues is not useful and decides to work on her own without any interruptions (i.e. no messaging). Figure 13 shows examples of monitor rules related to the need of receiving messages and not receiving messages for the pattern shown in Figure 12.

According to the monitor rules in Figure 13 the message feature is reactivated after a bank transaction finishes (confirm and logout) or not reactivated at all.

4. Related Work

Various approaches have been proposed to classify different context types when dealing with context-aware

systems. Some approaches consider the development of an application and propose context types concerned to physical sensing devices [14][16][21][27]. Other approaches focus on the formulation of general and hierarchical context sets [12][22][29] that are able to contain, according to a certain criteria, different context types. These classifications are formulated taking into account the different elements and components in a context-aware system. However, we have observed that in these approaches the user context types are either neglected or replaced by related context types such as location or time.

In [9], an ontology for context-aware pervasive computing environments is proposed. This ontology is centered on general concepts including people and places, and defines a set of properties and relationships associated with these general concepts. The main restriction in this proposal is that all the elements are defined according to a specific scenario and most of the identified user context types are related to physical attributes. Contrary, in [15][24] ontologies have been formulated considering the user as the main element. Similar to our approach, in these ontologies a central class represents the user profile and is associated to other classes concerned with other user characteristics such as skills or abilities. However, our ontology contains more specific user context types.

Different approaches have been proposed for monitoring service-based system. In [1][25] monitoring is performed by checking assumptions and conditions, which specify how services participate in a composition and the conditions the composition must satisfy. Assumptions and conditions are specified as behaviour properties in an expressive monitoring language. These behaviour properties are used by a monitor, to check the system operation, based on intercepted messages from the processes. A similar approach is taken in [20][32]. In this case, the correct behaviour is monitored by rules specified in EC. In this approach the rules are composed of assumptions and properties of a service-based system. The rules are used by a monitor to check the correctness of the system based on intercepted messages from the service-based system processes. In [2] monitoring is concerned with timeouts, runtime, and violation of functional contracts, which are described as monitoring rules. These monitoring rules are specified as comments in a BPEL process. When a service is invoked, its content is serialized, as an XML fragment, and sent together with the associated rules to another specially designed web service that acts as a monitor. All the above approaches use rules for monitoring the correct behaviour of a system. The rules, however, are formulated to monitor particular behaviors and processes and do not take into account changes in a service-based system, context characteristics of the user interacting with the system.

Some work to support dynamic selection of monitoring rules based on context information has been proposed in [13][33]. The approach in [33] uses monitor manager on top of existing monitoring tools to provide a policy driven interface for these tools. The policies describe how the monitoring infrastructure should react in the presence of changes. However, the rules used by the monitoring tools are not modified.

In [31] patterns for monitoring security properties such as confidentiality, integrity and availability have been proposed. Similar to our work, these patterns are expressed in EC language. Another approach that uses EC to represent patterns to support verification of physical interaction is proposed in [17]. In this work, patterns are prerequisites for an effective physical interaction. Events and fluents represent general actions and physical states (e.g., the location in which a user can have a physical interaction).

Although several approaches have been proposed for monitoring of service-based systems based on the use of monitor rules, none of these approaches consider the need for adaptation of the monitor rules.

5. Conclusions and Future Work

In this paper, we have presented a pattern-based approach for adaptation of monitor rules used by a monitor system. More specifically, in our approach, adaptation of monitor rules can be executed by (a) dynamic selection of the rules to be used, (b) automatic or semi-automatic modifications of monitor rules, and (c) automatic or semi-automatic creation of new monitor rules. The work is based on the use of patterns represented in Event Calculus for different user context types (e.g., roles, skills, preferences, needs, cognition, time, location, and environment). We have created an ontology for the different user context types to support representation of user models.

Currently, we are developing the monitor adaptor to support automatic identification of monitor rules, modifications of monitor rules, and creation of new monitor rules. We are also extending the set of patterns for the different user context types.

Acknowledgement

The work reported in this paper has been funded by the European Community's 7th Framework Programme under the Network of Excellence S-Cube – Grant Agreement no. 215483.

References

[1] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti, "Run-Time Monitoring of Instances and Classes of Web Service Compositions," in IEEE International Conference on Web Services (ICWS 2006), 2006, pp. 63–71.

[2] L. Baresi and S. Guinea, "Towards Dynamic Monitoring of WS-BPEL Processes," in Service-Oriented Computing - ICSOC 2005, Third International Conference, 2005, pp. 269–282.

[3] S. Bebbernou, L. Cavallaro, M.S. Hacid, R. Kazhamiakin, G. Kecskemeti, J.L Poizat, F. Silvestri, M. Uhlig, B. Wetzstein, " State of the Art Report, Gap Analysis of Knowledge on Principles, Techniques, and Methodologies for Monitoring and Adaptation of SBAs, S-CUBE Deliverable PO-JRA-1.2.1.

[4] C. Betini, D. Maggiorini, and D. Riboni. "Distributed Context Monitoring for the Adaptation of Continuous Services", In World Wide Web Journal (WWWJ), Special Issue on Multichannel Adaptive Information Systems on World Wide Web. Springer, 2007.

[5] D. Bianculli and C. Ghezzi, "Monitoring Conversational Web Services," in IW-SOSWE'07, 2007.

[6] A. Brown and M. Ryan, “Context-aware Monitoring of Untrusted Mobile Applications”, Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec 2009, Turin, Italy, June 3-5, 2009, LCN

[7] G. Calvary, J. Coutaz, D. Thévenin: “Embedding plasticity in the development process of interactive systems”; Proc. 6th Workshop User Interfaces for All, Bristol (2000).

[8] A.T.S. Chan and S.N. Chuang. “MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing”, IEEE Transactions on Software Engineering, vol. 29, n. 12, 2003.

[9] H. Chen, T. Finin, A. Joshi. An Ontology for Context-Aware Pervasive Computing Environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, May 2004.

[10] G. Chen and D. Kotz. “A Survey of Context-Aware Mobile Computing Research”, Dartmouth College, 2000.

[11] M. Comuzzi and G. Spanoudakis (2009) "Describing and Verifying Monitoring capabilities for SLA-driven Service-Based Systems" Proc. CAiSE Forum 2009, Amsterdam, the Netherlands.

[12] J.L. Crowley, J. Coutaz, G. Rey, P. Reignier, "Perceptual Components for Context Aware Computing", UBICOMP 2002, International Conference on Ubiquitous Computing, Goteborg, Sweden, September 2002.

[13] A-M. Dery-Pinna, J. Fierstone, and E. Picard. Component model and programming: A first step to manage human computer interaction adaptation. In Lecture Notes in Computer Science, pages 456–460. Springer-Verlag, 1999.

[14] A.K. Dey and G.D. Abowd. “The Context Toolkit: Aiding the Development of Context-Aware Applications, In Proceedings of the Workshop on Software Engineering for Wearable and Pervasive Computing, June, 2000.

[15] M. Golemati, A. Katifori, C. Vassilakis, G. Lepouras, C. Halatsis, Creating an Ontology for the User Profile: Method and Applications, Proceedings of the First IEEE International Conference on Research Challenges in Information Science (RCIS), Morocco 2007.

[16] G. Hackmann, C. Julien, J. Payton, and Gruia-Catalin Roman. Supporting generalized context interactions. In Proceedings of the Workshop on Software Engineering and Middleware (SEM 2004), 2004.

[17] F. Ishikawa, B. Suleiman, K. Yamamoto, and S. Honiden, 2009. Physical interaction in pervasive computing: formal modeling, analysis and verification. In Proceedings of the 2009 international Conference on Pervasive Services (London, United Kingdom, July 13 - 17, 2009). ICPS '09. ACM, New York, NY, 133-140.

[18] A. Keller and H. Ludwig, “The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services,” J. Network Syst. Manage., vol. 11, no. 1, 2003.

[19] H. Ludwig, A. Dan, and R. Kearney, “Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements,” in Service-Oriented Computing - ICSOC 2004, Second International Conference, 2004, pp. 65–74.

[20] K. Mahbub, G. Spanoudakis. Monitoring WS Agreements: An Event Calculus Based Approach, Test and Analysis of Service Oriented Systems, (eds) L. Baresi, E. di Nitto, Springer Verlag, 2007.

[21] R. Mayrhofer, H. Radi, and A. Ferscha. Recognizing and predicting context by learning from user behavior. Radiomatics: Journal of Communication Engineering, special issue on Advances in Mobile Multimedia, 1(1), May 2004.

[22] K. Mitchell, “Supporting the Development of Mobile Context-Aware Computing”, Ph.D. Thesis, Department of Computing, Lancaster University, January 2002.

[23] O. Moser, F. Rosenberg, and S. Dustdar, Non-intrusive monitoring and service adaptation for WS-BPEL, Proc. WWW 2008.

[24] I. Nébel, B. Smith, R. Paschke. A user profiling component with the aid of user ontologies. In: Proceedings of workshop learning - teaching - knowledge - adaptivity. Karlsruhe; 2003.

[25] M. Pistore and P. Traverso, “Assumption-Based Composition and Monitoring of Web Services,” in Test and Analysis of Web Services, L. Baresi and E. D. Nitto, Eds. Springer, 2007, pp. 307–335.

[26] Protégé ontology editor and knowledge acquisition system. http://protege.stanford.edu/

[27] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen. Contextphone: A prototyping platform for context-aware mobile applications. IEEE Pervasive Computing, 4(2):51–59, 2005.

[28] A. Sahai, V. Machiraju, M. Sayal, A. P. A. van Moorsel, and F. Casati, “Automated SLA Monitoring for Web Services,” in 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2002, 2002, pp. 28–41.

[29] A. Schmidt. “Ubiquitous Computing – Computing in Context”, PhD Thesis, University of Lancaster, 2002.

[30] M. Shanahan. "The event calculus explained", In Artificial Intelligence Today, LNCS: 1600, 409-430, Springer, 1999.

[31] G. Spanoudakis, C. Kloukinas, and K. Androutsopoulos, 2007. Towards security monitoring patterns. In Proceedings of the 2007 ACM Symposium on Applied Computing (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY, 1518-1525.

[32] G. Spanoudakis, K. Mahbub. Non Intrusive Monitoring of Service Based Systems, International Journal of Cooperative Information Systems, 15 (3), pp. 325-358, 2006.

[33] V. Talwar, C. Shankar, S. Rafaeli, D. Milojicic, S. Iyer, K. Farkas, and Y. Chen. Adaptive monitoring: Automated change management for monitoring systems. In Proceedings of the 13th Workshop of the HP OpenView University Association (HP-OVUA 2006) pp. 21-24, 2006.

Identifying, Modifying, Creating, and Removing Monitor Rules for Service Oriented Computing

Ricardo Contreras
Department of Computing
City University, London, Northampton Square
London EC1V 0HB, UK
+44 20 7040 8552
Ricardo.Contreras.1@soi.city.ac.uk

Andrea Zisman
Department of Computing
City University, London, Northampton Square
London EC1V 0HB, UK
+44 20 7040 8346
a.zisman@soi.city.ac.uk

ABSTRACT
Monitoring of service-based systems is considered an important activity to support service-oriented computing. Monitoring can be used to verify the behaviour of a service-based system, and the quality and contextual aspects of the services participating in the system. Existing approaches for monitoring service-based systems assume that monitor rules are pre-defined and known in advance, which is not always the case. In this paper, we present a pattern-based HCI-aware monitor adaptation framework to support identification, modification, creation, and removal of monitor rules. In the framework, changes in the monitor rules are based on user's interaction with a service-based system and different types of user context such as role, skill, cognition, need, and preferences. A prototype tool has been implemented to demonstrate the framework.

Keywords service monitoring, rules, patterns, adaptation, HCI context

1. INTRODUCTION
Service-oriented computing (SOC) has been recognized as an important paradigm for software development. Various approaches and techniques have been proposed to support different areas and activities related to SOC. One of these activities is concerned with monitoring of service-based systems; i.e., the activity of collecting information about the execution of a service-based system and verifying if the system is operating correctly by comparing the collected information with the properties of the system. These properties are known as *monitor properties* or *monitor rules* and can be used to verify the behaviour of a service-based system [3][4][24][29], the quality of the services participating in a system [11][17], and the contextual information of the services participating in the system and the system itself [6][8].

Existing approaches for monitoring service-based systems assume that monitor rules are pre-defined and known in advance. However, this is not always the case given that during execution time of a service-based system it is possible to have changes in the (i) service-based system or set of services used by the system (due to unavailability or malfunctioning of a service), (ii) types of interaction of the users with the system, and (iii) context characteristics of the user interacting with the system. Therefore, it is necessary to have ways of identifying monitor rules, modifying existing rules, removing existing rules that are obsolete, or creating new rules to support the needs of the monitor. We call this process *monitor adaptation.*

As an example, consider a Cultural Event service-based system (CE_SBS) that provides general information about cultural events in a certain city, allows ticket acquisition by different types of users, supports scheduling of different cultural programs in a year, provides catering services for certain special performances (e.g., premier performances, group performances), and allows both customers and professional critics to review cultural events. Suppose that the CE_SBS is used by managers and employees of several venues in a City (e.g., theaters, show houses, concert halls, opera houses), and by customers interested in cultural events that may or not be members of one or several of the venues. For example, managers of different venues (role) use this application to schedule and organize the different programs in a year for his/her venue taking into consideration other performances for that year in different venues; while a member of a venue (role) uses the system to periodically receive information about the different events in the venue based on the member's interest (preferences) and has priority on purchasing tickets before customers that are not members (case (iii) above). In these cases, it is necessary to have monitor rules for the different services used in the system depending on the user context. In addition, customers can book special events that include or not catering services (case (ii) above). In this case, new monitor rules relevant to booking and paying for catering services are required (needs). Moreover, general customers are able to provide simple reviews about a performance that they have seen (skills), while professional critics are supposed to provide reviews of a performance considering several detailed characteristics (skills). Furthermore, suppose the situation that after a while, the service in the system that verifies conflicts and constraints about various events in a City that is used to support the scheduling of events, becomes unavailable. Assume that a new service that provides these functionalities, but also allows searching and allocating necessary physical and personnel resources for the various events is used to replace the initial service. In this case, the monitor system needs to verify the new functionality provided by the service (i.e., resource allocation) (case (i) above).

In this paper we present the MADap (Monitor ADaptation) framework for monitor adaptation as defined above. The work presented in this paper has been carried out as part of the European funded Network of Excellence S-Cube [25]. Our framework concentrates on HCI-aware monitor adaptation in which changes in the monitor rules are based on user's interaction with a service-based system and different types of user context. The user context information of our concern includes *cognition*, *role*, *skill*, *need*, and *preferences* of a user, represented in user models based on an ontology that we have created. The framework is based on the use of patterns for the monitor rules representing different context types. The monitor rules are concerned with the execution parts of a service-based system specification for the user context types. The patterns are used to support the (a) identification of monitor

rules, (b) modification of monitor rules, (c) creation of new monitor rules, and (d) removal of obsolete monitor rules.

The remainder of this paper is structured as follows. Section 2 describes an overview of our framework. Section 3 presents the monitor adaptation process used by the framework, including the processes to identify, modify, create, and remove monitor rules. Section 4 presents implementation aspects and initial evaluation of the framework. Section 5 discusses existing related work. Section 6 summarizes the work and describes future work.

2. FRAMEWORK OVERVIEW

The MADap framework assumes two different types of user context, following the classification proposed in [10][13][19], namely (a) *direct user context types* and (b) *related user context types*. The direct user context types represent information of the characteristics of the users and include *role*, *skill*, *need*, *preferences*, and *cognition* context types. The related user context types represent information that may influence user information and include *time*, *location*, and *environment* context types. We have developed an ontology to represent the different user context types. A brief description of the various user context types is presented in Table 1.

Table 1. Description of user context types

Context Type		Description
Direct	Role	*The social behavior of an individual within the domain of a service-based system*
	Skill	*The level of expertise of an individual with respect to a service-based system*
	Need	*An individual's requirement or desire from a service-based system*
	Preferences	*An individual's choice over pre-established alternatives of computational resources of a service-based system*
	Cognition	*Individual's characteristics associated with the process of thought (the ways that individuals think, feel or react)*
Related	Time	*The moment an individual interacts with a service-based system*
	Location	*The place where an individual interacts with a service-based system*
	Environment	*Information related to the environment where a service-based system is used*

A graphical representation of the ontology we have developed is shown in Figure 1. As shown in Figure 1, the different context types are represented as classes, with subclasses in some cases, and are associated with a central class representing the user. These associations represent relationships between the different attributes of a context type (e.g., occupation for context class role) and a user class. For each class their attributes and respective data types are presented inside the class. Please note that at this stage we do not consider associations between the different context types given that the focus of the work is on relationships between users and contexts and not between context and context.

As described in Section 1, the framework is based on the use of *rule patterns* for the different user context types. It is possible to have different patterns for the same context type. Both the patterns and the monitor rules are described in Event-Calculus (EC) [26]. The use of Event Calculus (EC) to describe monitor rules has been advocated in [11][29] and has shown to be appropriate to support the representation of several types of rules.

More specifically, Event Calculus allows (i) rules to be represented as first order logic, which provides sufficient expressiveness for a large range of applications; (ii) specification of quantitative temporal constraints and relationships that are necessary to be taken into consideration when monitoring service-based systems; (iii) distinction between events and states that are necessary to describe the behavior of a system and interaction of users with the system; and (iv) definition of the influences between events and states.

Event Calculus uses *events* and *fluents* to represent the behavior of a system. An event occurs at a specific instance of time and may change the state of a system. A fluent is a condition of a system state and may be affected by the occurrences of events. Both events and fluents are represented in EC by predicates.

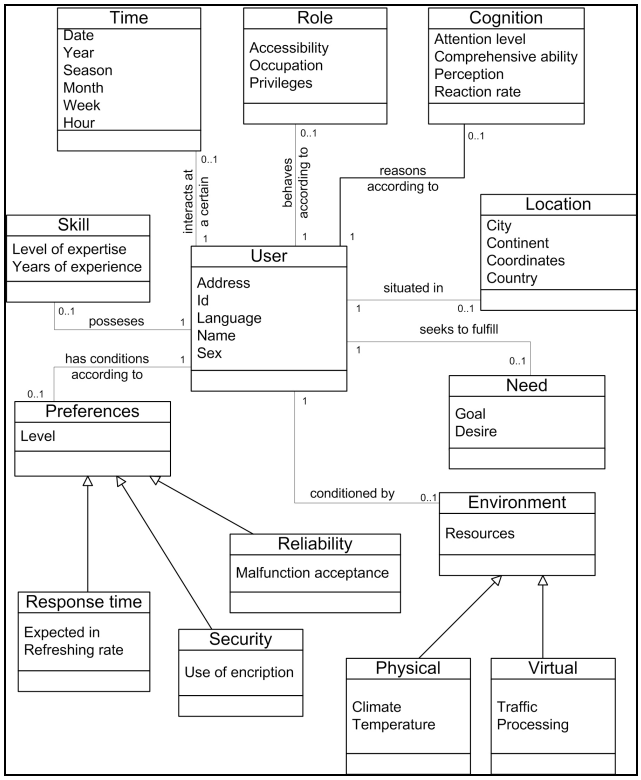


Figure 1: User context ontology

The occurrence of an *event*, at some time *t*, is represented by the predicate *Happens(event,t,R(t1,t2))*, which means an *event* occurs at a time *t*, where *t* is within an interval between *t1* and *t2*. The time boundaries, represented by *t1* and *t2,* can be specified using time variables or arithmetic expressions over time variables, and represent the lower and upper time boundaries. The initialization of a *fluent*, is represented by the predicate *Initiates(event,fluent,t)*, which means a *fluent* starts to holds after an *event* occurs at a time *t*. The predicate *HoldsAt(fluent,t)*, means a *fluent* holds (is valid) at a time *t*. The termination of a *fluent*, is represented by the predicate *Terminates(event,fluent,t)*, which means a *fluent* ceases to hold after an *event* occurs at a time *t*. A detailed description of Event Calculus is out of the scope of this paper [26].

A rule pattern is composed of two parts, namely (a*) monitor rule* part and (b) *assumptions* part. The monitor rules represent properties of a service-based system that need to be monitored. The assumptions represent event calculus formulae that need to be used to identify state information of the system. An example of a

rule pattern for role context type and the instantiation of this pattern for the CE_SBS described in Section 1 for role "Manager" are shown in Figures 2.a and 2.b, respectively.

Monitor rule

Happens (ic_Initial-Event, t1, R(t1,t1) => **Happens** (ic_Event, t2, R(t1,tn))

Assumption

Happens (ic_Event, t, R(t, t)) => **Initiates** (ic_Event, *fluent*, t)

Figure 2.a: Rule pattern for role context type

Monitor Rule

Happens (*ic_start_CES*, t1, R(t1,t1)) =>

Happens (*ic_performance_schedule*, t2, R(t1,t1+2500))

Assumption

Happens (*ic_performance_schedule*, t1, R(t1,t1)) =>

Initiates (*ic_performance_schedule*, *per-schedule*, t1)

Figure 2.b: Instantiation of rule pattern in Figure 2.a

The monitor rule part in Figures 2.a and 2.b state that after the initial event of the service-based system specification (*ic_Initial-Event* and its instantiation *ic_Start_CES*) is executed, a system event concerned with a user context role (*ic_Event* and its instantiation *ic_performance_schedule*) should be executed in a time t2, where t1 ≤ t2 ≤ tn (t1 ≤ t2 ≤ t1+2500). The assumption part states that when different system events occur, different fluents are initiated. In this case, when event *ic_performance_schedule* occurs, fluent *per-schedule* is instantiated. The patterns used by the framework are general in order to be employed in different types of service-based systems. Events in a pattern can represent either requests for an operation (when specified with *ic* prefix) or responses from an operation (when specified with *ir* prefix).

A rule pattern may have *invariant parts*, which depend on the context type associated with the pattern. An invariant part does not change for distinct instantiations of the pattern. The invariant parts for the role context type pattern in Figure 2.a is shown in Figure 3 (variant parts are represented by ____). Other patterns have been created to represent the other context types and can be found in [20]. They are not shown here due to space limitations.

Monitor rule

Happens (ic_Initial-Event, t1, R(t1,t1) =>

Happens (____, t2, R(t1,tn))

Assumption

Happens (____, t, R(t, t)) => **Initiates** (____, ____ , t)

Figure 3: Example of invariant part for role type pattern

Figure 4 presents the architectural overview of the MADap framework. As shown in the figure, the main components of the framework are *Rule Adaptor*, *Path Identifier*, *Rule Verifier*, and *Monitor*. The framework also uses *Rule Patterns*, *Semi-instantiated Patterns*, *Monitor Rules*, *User Models, Service-based System (SBS) Specification,* and *Service Level Agreements (SLAs)*. It assumes rule patterns and monitor rules in a repository.

The Rule Adaptor is responsible for the identification, modification, creation, and removal of monitor rules. More specifically, it receives events about changes in the context characteristics of the user or interaction of the users in the system, and invokes the Path Identifier to identify paths in the specification of the service-based system that are relevant to the event. The Path Identifier retrieves the parts in the specification that are related to the context type represented in the event and its instance (e.g., context *role*, instance *manager*).

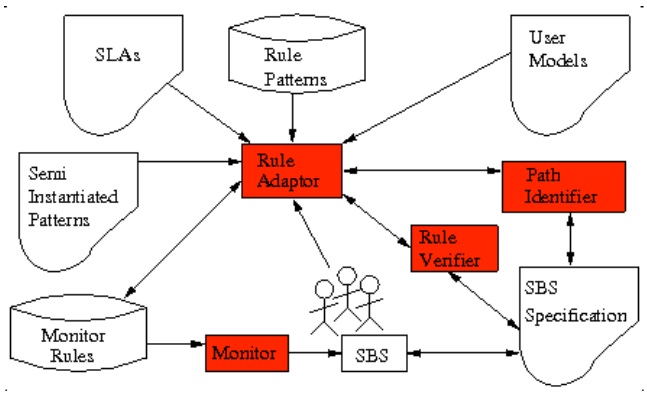


Figure 4. Architecture overview of MADap framework

The Rule Adaptor also uses the context type to identify the relevant Rule Patterns for this context type and instantiates these patterns with the identified information from the service-based system specification and User Models. These are called *semi-instantiated rule patterns*. The User Models represent the characteristics of the users.

The Rule Verifier is used to verify if an existing monitor rule in the repository is still a valid rule for a service-based system. It is possible to have rules that become obsolete due to changes in the service-based system (e.g., certain functionalities are not executed anymore, new functionalities are added to the system).

We assume specifications of service-based systems in BPEL [7] due to its wide acceptance, and that the BPEL processes have conditions on the various user context types and their instances. This assumption is not unrealistic given that the different behavior of a service-based system due to different user characteristics needs to be represented in the service-based system specification.

The Rule Adaptor uses the semi-instantiated rule patterns to identify monitor rules. In the case where monitor rules that totally match the semi-instantiated rule patterns are identified, these rules are either used as they stand by the Monitor component or have their time values modified, when necessary, and subsequently used by the Monitor component. In the situation in which no rules that match the semi-instantiated rule patterns are identified, new monitor rules are created based on the semi-instantiated patterns. In the case in which there are monitor rules that match the invariant parts of the semi-instantiated rule patterns, the Rule Verifier checks if these rules are still valid for the service-based system. In positive case, these rules have their time values modified, if necessary. Otherwise, these rules are removed from the repository and a new rule based on the semi-instantiated pattern is created. The newly created rules are added into the repository and used by the Monitor component to verify the service-based system.

In the framework, we use the Monitor tool described in [29]. However, our approach can be used with any monitor tool that makes use of monitor rules represented in Event Calculus. The Monitor tool receives requests from a service requestor to verify, at regular intervals, the satisfiability of properties (represented as monitor rules) of a service-based system. It intercepts run-time

messages exchanged between a service-based system and its services and verifies the satisfiability of the properties against these messages. It contains (a) a *service client* that is responsible to invoke a service in a service-based system; (b) and *event collector* that is responsible to gather information during the execution of a service-based system and the services deployed by the service based system, or information exchanged between the service client and its respective services; and (c) an *analyzer* that is responsible to check the satisfiability of the properties.

3. MONITOR ADAPTATION PROCESS

In the framework, the monitor adaptation process is triggered by an event representing a context type Ci. Based on the context type, the Rule Adaptor identifies the patterns concerned with context Ci, invokes the Path Identifier to identify the parts of the BPEL specification that are related to Ci, and uses this information to semi-instantiate the identified patterns. The semi-instantiated patterns are compared to existing rules in the repository in order to identify if a relevant rule (a) already exists in the repository and (a.1) can be used as it stands, (a.2) needs to be modified, (a.3) needs to be removed, or (b) needs to be created.

The need to use information from the service-based system specification to semi-instantiate the relevant pattern is due to the fact that in some situations the pattern itself is not sufficient to support the identification of the correct monitor rule in the repository. For example, suppose an event for *customer* role context type. Assume the pattern for context type role shown in Figure 2.a and the monitor rules in Figure 5 below in the rule repository (without the assumptions). In this case, all the three rules in Figure 5 match the rule pattern in Figure 2.a. However, rules R1 and R2 are concerned with role *customer* while rule R3 is concerned with role *manager*. If the pattern is used to identify rules in the repository, all three rules will be returned. But, following the part of the BPEL specification for the CE_SBS shown in Figure 6, the events representing the operations *shows* and *timetables* are relevant to role *customer*. Using these events, the two semi-instantiated patterns shown in Figure 7 will be specified and rules R1 and R2 will be correctly matched with the semi-instantiated patterns. It should be noted that information about these events could not be part of the initial patterns since the patterns are general for a certain context type and the events and the monitor rules are specific to a service-based system and instances of the various context types.

R1 **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
Happens (*ic_shows*, t2, R(t1,t1+2000))
R2 **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
Happens (*ic_timetables*, t2, R(t1,t1+4000))
R3 **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
Happens (*ic_performanceSchedule*, t2, R(t1,t1+2500))

Figure 5. Examples of monitor rules for role context type

As shown in Figure 7, the semi-instantiated patterns do not have the values for time variables or time gaps. In order to specify the boundary of the values for the times, the framework assumes these values to be identified from the response time information of a service, or operations of a service, that are normally defined in Service Level Agreements (SLAs) between the services participating in the service-based system and the system itself. Another option is to use historical execution time data for a service, when available. The assumption that SLAs will be available for participating services is not unrealistic, since SLAs are currently used to establish business agreements between service providers and consumers. Moreover, the response times of a service or operations are attributes that appear in SLAs.

```
…<bpel:condition>
<![CDATA[$input.payload/tns:role_occupation="Customer"]]>
</bpel:condition>
<bpel:invoke name="Shows" partnerLink="Shows"
operation="shows" portType="ns:Shows"
inputVariable="ShowsRequest"
   outputVariable="ShowsResponse"> </bpel:invoke>
<bpel:invoke name="TimeTables" partnerLink="TimeTables"
   operation="timeTables" portType="ns:TimeTables"
   inputVariable="TimeTablesRequest"
outputVariable="TimeTablesResponse">
</bpel:invoke> …
```

Figure 6. Part of the BPEL process for CE_SBS

SI_RP1: **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
Happens (*ic_shows*, t2, R(t1,tn))
SI_RP2: **Happens** (*ic_start CES*, t1, R(t1,t1)) =>
Happens (*ic_timetables*, t2, R(t1,tn))

Figure 7. Examples of semi-instantiated role patterns

Figure 8 presents an algorithm in pseudo-code for the monitor adaptation process used in the framework. As shown in Figure 8, the process consists of searching in the repository for monitor rules that match the semi-instantiated pattern. In the case that there are rules that fully match the semi-instantiated pattern, the process verifies if the time values in the rules are consistent with the response times of the SLAs for the respective operations and services. In positive case, the rules are maintained in the repository. Otherwise, the rules are modified with new time values according to the information in the SLAs.

In the case in which there are rules in the repository that only match the invariant parts of the semi-instantiated pattern, the time values of the patterns are instantiated based on information for SLAs and new rules are created (fully-instantiated patterns). The process verifies for every identified rule in the repository that matches the semi-instantiated patterns, if the rule is a valid rule. This is done by traversing the service-based system specification and verifying if the information in the rule is still a valid path in the specification. In positive case, the time values for the rule are checked against related SLAs and adjusted if necessary. In negative case, the rule is removed from the repository. The new created rules are added in the repository.

In the case in which there are no monitor rules in the repository that match the semi-instantiated patterns, the time values are instantiated based on information from SLAs, new rules are created (fully-instantiated patterns), and added to the repository.

As an example, consider the CE_SBS described in Section 1. Suppose that an opera critic accesses the system to input a review for the performance of Carmen that is currently being presented at the London Royal Opera House. In this case, the skill pattern shown in Figure 9 is identified by the Rule Adaptor after receiving an event for context skill referring to *professional critic* (skillful), and the semi-instantiated pattern in Figure 10 is created based on information from the BPEL process. In Figure 9, the variant part of the pattern is shown in grey. Suppose rule R4

shown in Figure 11 in the repository that matches the invariant parts of the semi-instantiated pattern in Figure 10. Assume, that rule R4 is not a valid rule for the current version of the service-based system (operation *sceneryAdaptation* is not in the BPEL process anymore due to changes in the system). R4 is removed from the repository and rule R5 shown in Figure 12, with the time values identified from respective SLAs, is added to the repository.

```
Monitor_Adaptation (SI_Rule, SBS_Spec, SLAs, RRep) {
//SI_Rule: semi-instantiated pattern
//FI_Rule: fully-instantiated pattern
//SBS_Spec: service-based system specification
//SLAs: SLAs for the services and operations in the SI_Rule
//RRep: Rule Repository
//R: Rules in RRep
Search SI_Rule in RRep;
If (RRep has Rules that fully match SI_Rule) {
  For Every R in Rules {
    If (time in SLAs is within time values in R) {
      Do-nothing;}
    Else {Adjust time in R based on SLAs;}
    End If
  } End For
Else {
  Create FI_Rule by instantiating SI_Rule time with times in SLAs;
  If (RRep has Rules that only match invariant parts of SI_Rule) {
    For Every R in Rules {
      If (there is a path in SBS_Spec that uses R) {
        // Rule R is not obsolete
        If (time in SLAs is within time values in R) {
          Do-nothing; }
        Else {Adjust time in R based on SLAs; }
        End If
      Else {Remove R from RRep; }
      End If
    } End For
    Add FI_Rule to RRep; }
  Else {
   //There are no rules that match the semi-instantiated rule
    Create FI_Rule by instantiating SI_Rule time with times in SLAs;
    Add FI_Rule to RRep; }
  End If
} End Monitor_Adaptation
```

Figure 8. Algorithm for the monitor adaptation process

Monitor rule

Happens (ic_Initial-Event, t1, R(t1,t1) and

Happens (ir_Event-User-Op, t2, R(t1,tn_1)) …

=> **Happens** (ic_LEvent, tn, R(t1,tn_3)) tn > t2

Assumption

Happens (ic_LEvent, t1, R(t1, t1)) =>

Initiates (ic_LEvent, *fluent*, t1)

Figure 9: Example of skill context pattern

Monitor Rule

Happens (ic_start_CES, t1, R(t1,t1)) and

Happens (ir_artistsPerformance, t2, R(t1,tn_1)) and

Happens (ir_direction, t3, R(t1,tn_2))

=> **Happens** (ic_professionalReview, t4, R(t1,tn_3) t4>t2, t4>t3

Assumption

Happens (ic_professionalReview, t1, R(t1,t1))

=> **Initiates**(ic_ professionalReview, professionalReview, t1)

Figure 10: Example of semi-instantiated skill pattern

Monitor Rule

Happens (ic_start_CES, t1, R(t1,t1)) and

Happens (ir_artistsPerformance, t2, R(t1,t1+30001)) and

Happens (ir_sceneryAdaptation, t3, R(t1,t1+40002)) and

Happens (ir_direction, t4, R(t1,t1+50003)) =>

Happens (ic_profesionalReview, t5, R(t1,60004)

t5>t2, t5>t3, t5>t4

Assumption

Happens (ic_profesionalReview, t,R(t,t))

=> **Initiates** (ic_profesionalReview, profesionalReview, t)

Figure 11: Example of rule R4

Monitor Rule

Happens (ic_start_CES, t1, R(t1,t1)) and

Happens (ir_artistsPerformance, t2, R(t1,t1+30001)) and

Happens (ir_direction, t3, R(t1,t1+45003))

=> **Happens** (ic_profesionalReview, t4, R(t1,60004)

t4>t2, t4>t3

Assumption

Happens (ic_profesionalReview, t,R(t,t))

=> **Initiates**(ic_profesionalReview, profesionalReview, t)

Figure 12: Example of rule R5

4. IMPLEMENTATION ASPECTS AND EVALUATION

A prototype tool of the MADap framework has been implemented in Java. The tool can be used to adapt monitor rules specified in Event Calculus [26]. The prototype was designed to take as input an event representing one of the direct user context (see Table 1) types and deciding if a monitor rule could be identified, modified, created, or removed. The prototype tool was used to evaluate the framework for the four adaptation cases proposed in this paper.

We evaluated the framework in an extension of the CE_SBS described in Section 1 with seven services, namely: S1:Ticket Purchase Service, S2:Payment Service, S3:Performance Information Service, S4:Performance Scheduling Service, S5:Resource Search and Allocation Service, S6:Reviewing Services, and S7:Catering Services. The evaluation was conducted for each direct user context type (role, skill, need, preferences, cognition) in five different cases, as described below.

Case 1: *Empty rule repository*

This case was considered to verify the creation of new monitor rules and assumptions for each different context type. A total of 29 monitor rules and assumptions were created in the repository, broken down as follows:

8 rules/assumptions due to context type role,

6 rules/assumptions due to context type need,

4 rules/assumptions due to context type skill,

8 rules/assumptions due to context type cognition, and

3 rules/assumptions due to context type preferences.

The number of rules (and assumptions) for each context type is directly related to the different roles, needs, skills, cognition, and preference characteristics of the users of the service-based system, and the relevant functionalities of the system for the context types.

Case 2: *Rule repository with 100 rules not related to CE_SBS*
This case was also considered to verify the creation of new monitor rules for each different context type given that all the 100 unrelated rules did not match fully or partially (matching of the invariant parts) the semi-instantiated patterns. As in Case 1 above, 29 monitor rules were created in the repository with the same number of rules for each context type.

Case 3: *Rule repository with all 29 relevant rules for CE_SBS*
This case was considered to verify if monitor rules that can be used for monitoring a service-based system at different stages of its execution could be identified. For each of the different context types, the relevant rules where identified.

Case 4: *Rule repository with all 29 relevant rules for CE_SBS and with 100 rules not related to the system*
This case was also considered to verify if monitor rules that can be used for monitoring a service-based system at different stages of its execution could be identified. Despite the extra not related rules in the repository, as in Case 3 above, the relevant rules for each different context type were identified from the 29 relevant monitor rules in the repository.

Case 5: *Rule repository with 100 rules not related to the system, 27 rules relevant to the system, and 5 rules that match the invariant pattern parts for each context type*
In this case, the set of 27 relevant rules and 5 rules that match the invariant pattern parts are different for each context type. More specifically, for each context type, the 27 rules were selected from the set of 29 relevant rules by removing two rules for the specific context type. Given the mixture of the rules in the repository, in this case, the framework was able to identify the remaining rules for a certain context from the set of 27 relevant rules, to create the two rules that were initially removed from the set of relevant rules, and to identify 5 rules that match the invariant parts and verify if these rules were supposed to be removed from the repository or maintained.

This initial evaluation has demonstrated that the framework can adapt monitor rules for different user context types in the four proposed ways. It also shows that there is a direct correlation in the number of monitor rules and the different instances of the various context types for a service-based system. In addition, the approach supports the removal of monitor rules for those rules that become obsolete due to changes in the service-based system and the verification of this situation when trying to identify rules in the repository or create new rules in the repository. This allows maintaining the size of the repository with rules that are useful for monitoring a service-based system.

5. RELATED WORK

In [9], an ontology for context-aware pervasive computing environments is proposed. This ontology is centered on general concepts including people and places. However, in this proposal all the elements are defined according to a specific scenario and most of the identified user context types are related to physical attributes. Contrary, in [16][23] otologies have been formulated considering the user as the main element. Similar to our approach, in these ontologies a central class represents the user models and is associated to other classes concerned with other user characteristics such as skills or abilities. Our framework considers more specific types of user context.

Different approaches have been proposed for monitoring service-based systems. In [24] monitoring is performed by checking assumptions and conditions, which specify how services participate in a composition and the conditions the composition must satisfy. Assumptions and conditions are specified as behaviour properties in an expressive monitoring language. These behaviour properties are used by a monitor, to check the system operation, based on intercepted messages from the processes. A similar approach is taken in [29] where behaviour is monitored by rules specified in EC. In this approach the rules are composed of assumptions and properties of a service-based system. In [4] monitoring is concerned with timeouts, runtime, and violation of functional contracts, which are described as monitoring rules. These monitoring rules are specified as comments in a BPEL process. When a service is invoked, its content is serialized, as an XML fragment, and sent together with the associated rules to another specially designed web service that acts as a monitor. All the above approaches use rules for monitoring the correct behaviour of a system. The rules are formulated to monitor particular behaviors and do not consider changes in a service-based system and user context characteristics.

Some work to support dynamic selection of monitoring rules based on context information has been proposed in [14][30]. The approach in [30] uses monitor manager on top of existing monitoring tools to provide a policy driven interface for these tools. The policies describe how the monitoring infrastructure should react in the presence of changes. The rules used by the monitoring tools are not modified.

In [28] patterns for monitoring security properties such as confidentiality, integrity and availability have been proposed. Similar to our work, these patterns are expressed in EC language. Another approach that uses EC to represent patterns to support verification of physical interaction is proposed in [17]. In this work, patterns are prerequisites for an effective physical interaction. In [14] the authors propose the use of a pattern-based approach to support presentation, codification, and reuse of property specification for finite-state verification. This work is used in [27] where temporal logic patterns for runtime monitoring of web service conversations are used.

Recently, a few approaches that support adaptation of service-based systems have started to appear [1][2][5][21][22]. The work in [5] proposes an approach towards self-healing for services compositions based on monitoring rules and reaction strategies. Another approach for self-healing is found in the PAWS framework [1] in which monitor and recovery actions are used. In [2] the authors present a context-aware adaptive service approach. The VieDAME framework [22] uses an aspect-oriented approach to allow adaptation of service-based systems for certain QoS criteria based on various alternative services. The work in [21] is based on augmenting service monitoring with online testing to identify possible failures in the system.

Although several approaches have been proposed for monitoring and adaptation of service-based systems, none of these approaches consider the need for adaptation of the monitor rules.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a monitor adaptation framework (MADap) that supports the identification, modification, creation, and removal of monitor rules due to changes in the (i) service-based system or set of services used in the system, (ii) types of user interaction with the system, and (iii) context characteristics of the user interacting with the system. The framework considers different types of user context such as role, skill, need, preferences, and cognition. MADap is based on the use of monitor rule patterns for each different user context type. The approach assumes monitor rules, assumptions, and patterns specified in

Event Calculus [26]. A prototype tool has been developed to illustrate and evaluate the framework. The evaluation has demonstrated that the framework can identify, modify, create, or remove obsolete monitor rules when necessary.

We are currently extending the set of patterns for the different types of user context. We are also conducting other evaluation of the framework for more complex service-based systems. Moreover, we are analyzing the performance of the framework and how a monitor component could use the created, identified, and modified rules during run-time.

7. ACKNOWLEDGMENTS

The work reported in this paper has been funded by the European Community's 7th Framework Programme under the Network of Excellence S-Cube – Grant Agreement no. 215483.

8. REFERENCES

[1] Ardagna, D., Comuzzi M., Mussi, E., Pernici, B., Plebani, P. "PAWS: A Framework for Executing Adaptive Web-Service Processes". IEEE Software, 24 (6), 2007.

[2] Autili, M., Di Benedetto, P. and Iverardi. "Context-aware Adaptive Services: The PLASTIC Approach". Proc. of the 12th International Conference on Fundamental Approaches to Software Engineering, FASE, 2009.

[3] Barbon, F., Traverso, P., Pistore, M. and Trainotti, M., "Run-Time Monitoring of Instances and Classes of Web Service Compositions," in IEEE International Conference on Web Services (ICWS 2006), 2006.

[4] Baresi, L. and Guinea, S., "Towards Dynamic Monitoring of WS-BPEL Processes. Third International Conference on Service Oriented Computing, 2005.

[5] Baresi, L., Ghezzi, C., Guinea, S., "Towards Self-Healing Compositions of Services. Studies in Computational Intelligence", v. 42, Springer (2007).

[6] Betini, C., Maggiorini, D. and Riboni, D., "Distributed Context Monitoring for the Adaptation of Continuous Services", In WWW Journal, Special Issue on Multichannel Adaptive Information Systems on WWW. Springer, 2007.

[7] BPEL4WS.http://www128.ibm.com/developerworks/library/specification/ws-bpel/

[8] Brown A. and Ryan, M., "Context-aware Monitoring of Untrusted Mobile Applications", Security and Privacy in Mobile Information and Communication Systems, First International ICST Conference, MobiSec, Italy, June 2009.

[9] Chen, H., Finin, T., Joshi, A., An Ontology for Context-Aware Pervasive Computing Environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, May 2004.

[10] Chen G. and Kotz, D., "A Survey of Context-Aware Mobile Computing Research", Dartmouth College, 2000.

[11] Comuzzi, M. and Spanoudakis, G., "Describing and Verifying Monitoring capabilities for SLA-driven Service-Based System", Proc. CAiSE Forum 2009, Amsterdam.

[12] Dery-Pinna, A-M., Fierstone, J. and Picard, E., Component model and programming: A first step to manage human computer interaction adaptation. In Lecture Notes in Computer Science, Springer-Verlag, 1999.

[13] Dey A.K. and Abowd, G.D., "The Context Toolkit: Aiding the Development of Context-Aware Applications, In Proceedings of the Workshop on Software Engineering for Wearable and Pervasive Computing, June, 2000.

[14] Dwyer, M.B., Avrunin, G.S. and Corbett, J.C., "Patterns in Property Specifications for Finite-state Verification". 31st International Conference on Software Engineering, 1999.

[15] Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G., "Model evolution by run-time parameter adaptation." Proc on the 31st International Conference on Software Engineering, Vancouver, Canada, 2009.

[16] Golemati, M., Katifori, A., Vassilakis, C., Lepouras, G., Halatsis, C., Creating an Ontology for the User Profile: Method and Applications, Proceedings of the First IEEE International Conference on Research Challenges in Information Science, 2007.

[17] Ishikawa, F., Suleiman, B., Yamamoto, K. and Honiden, S. Physical interaction in pervasive computing: formal modeling, analysis and verification. International Conference on Pervasive Services, 2009, London, UK.

[18] Ludwig, H., Dan, A. and Kearney, R., "Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements," in Service-Oriented Computing - ICSOC 2004, Second International Conference, 2004, pp. 65–74.

[19] Maiden N., editor. Codified Human-Computer Interaction (HCI) Knowledge and Context Factors. S-Cube project deliverable: PO-JRA-1.1.3 http://www.s-cube-network.eu/achievements-results/s-cube-deliverables.

[20] MADap: Monitor Adaptation Project. http://vega.soi.city.ac.uk/~abdw747/MADap

[21] Metzger, A., Pohl, K., Sammodi, O., Rzepka, M., "Towards Proactive Adaptation with Confidence – Augmenting Service Monitoring with Online Testing. Workshop on Software Engineering for Adaptive and Self-Managing Systems, South Africa, 2010.

[22] Moser, O., Rosenberg, F. and Dustdar, S., Non-intrusive monitoring and service adaptation for WS-BPEL, Proc. WWW 2008.

[23] Nébel, I., Smith, B., Paschke, R., A user profiling component with the aid of user ontologies. Proc. of workshop learning teaching knowledge adaptivity, Karlsruhe; 2003.

[24] Pistore M. and Traverso P. Assumption-Based Composition and Monitoring of Web Services, Test and Analysis of Web Services, L. Baresi and E. D. Nitto Eds., 2007.

[25] SCube. Software Services and Systems Network of Excellence. http://www.s-cube-network.eu/

[26] Shanahan, M., "The event calculus explained", In Artificial Intelligence Today, LNCS: 1600, 409-430, Springer, 1999.

[27] Simmonds, J., Chechik, M., Nejati, S., Litani, E., O'Farrel, B., "Property Patterns for Runtime Monitoring of Web Service Conversations". RV 2008: 137-157.

[28] Spanoudakis, G., Kloukinas, C. and Androutsopoulos, K., 2007. Towards security monitoring patterns. In Proceedings of the 2007 ACM Symposium on Applied Computing, SAC, 2007. ACM, New York, NY, 1518-1525.

[29] Spanoudakis, G., Mahbub, K., Non Intrusive Monitoring of Service Based Systems, International Journal of Cooperative Information Systems, 15 (3), pp. 325-358, 2006.

[30] Talwar, V., Shankar, C., Rafaeli, S., Milojicic, D., Iyer, S., Farkas, K. and Chen, Y., Adaptive monitoring: Automated change management for monitoring systems. 13th Workshop of the HP OpenView University Association 2006.

A Context-driven Adaptation Process for Service-based Applications*

Antonio Bucchiarone and Raman Kazhamiakin*
Cinzia Cappiello, Elisabetta di Nitto and Valentina Mazza
*FBK-IRST, Trento, Italy
{bucchiarone,raman}@fbk.eu
Politecnico di Milano, Italy
{cappiell,dinitto,vmazza}@elet.polimi.it

ABSTRACT

When building open systems the evolution of requirements and context is the norm rather than the exception. Therefore, it is important to make sure that the system is able to evolve as well without necessarily starting a completely new development process, and possibly on the fly. In this paper we specifically focus on the role of the *context* in the adaptation activities. For us context has various different facets as it includes information ranging from the situation in which users exploit a service-based application to the conditions under which the component services can be exploited. We elaborate on how and when the context should be defined, exploited, and evolved, and on the impact it has on the various activities related to adaptation of service-based applications. We use a case study to exemplify our first findings on this subject.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*Methodologies*

Keywords

Service-oriented Systems, Context-awareness, Adaptation

1. INTRODUCTION

Traditional software systems are usually designed in order to address a specific set of requirements within a specific execution context. Even though the evolution of requirements and execution context is considered possible, it is assumed that it will have an impact on the new versions of the software system, not on the one that is currently under operation. This assumption is not anymore true when we consider those open systems that are built by composing existing services available on the network. In these cases we should consider the evolution of requirements and context as the norm rather than the exception, and, as such, we should make sure that the system is able to evolve as well without necessarily starting a completely new development process, and possibly on the fly. As we discuss in the related work section (see Section 2), the literature offers technical approaches to manage the on the fly adaptation of service-based applications. However, to our knowledge, a comprehensive approach to design and develop adaptable Service-Based Applications (SBAs) is still missing. Our work tries to fill this gap. In this paper we specifically focus on the role of the *context* in the adaptation activities. For us context has various different facets as it includes information ranging from the situation in which users exploit a service-based application to the conditions under which the component services can be exploited. In general, the identification of the aspects that should be part of the context depends on the specific application and should be performed very early. In fact, starting from the requirement analysis phase where, in parallel to the precise definition of requirements, a proper *context model* has to be defined. This context model is the basis for the definition of those situations that trigger the adaptation or evolution of a service-based application, and, at runtime, enables the identification and the collection of the proper context information. Of course, as any other software artifact, this context model is not fixed once for all but can evolve together with the application, and therefore its evolution has to be managed and kept under control as well.

In this paper we start studying these aspects in detail and exemplify them through a case study. Consistently, the paper is structured as follows. Section 2 discusses previous contributions that dealt with context-aware applications and adaptation needs and describes the main open issues that are addressed in this paper. Section 3 identifies the additional requirements and phases that should be considered in the design life-cycle of an adaptive SBA while the context elements considered as relevant in the adaptation process are detailed and modelled in Section 4. A contex-driven adaptation process is proposed in Section 5. Finally, Section 6 presents the instantiation of the proposed process on a case study in the e-government domain.

2. RELATED WORK

In many SBAs the role of the context is fundamental in realizing the adaptation functionalities [4]. Indeed, contextual

*The research leading to these results has received funding from the European Community Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

SIGMOD 2006, June 27–29, 2006, Chicago, Illinois, USA.

changes are often the key factors that entail the SBA adaptation and somehow drive the way the adaptation is performed. [14] contains an analysis of the context factors in context-aware computing and in software engineering. The knowledge of the computing environment (network characteristics,resources), of the user (profile, location, social situation), of the physical parameters (all the measurable properties of the environment such as the noise level, the pressure, the temperature) contribute to the context knowledge. This information is often codified in a so called *context model*.

The literature is rich of proposals for context models. In [15] context representations are classified as *Key-value models* (key-value pairs are used to represent the features of a system), *Mark-up scheme models* (a hierarchical structure for context representation), *Graphical models* (general purpose modeling instruments are used to represent context information), *Object Oriented Models* (each context information is represented by a different object and data can be accessed by defined interfaces), *Logic Based models* (context is expressed in terms of facts, expressions, and rules) and, finally, *Ontology Based models* (information about real world is represented using data structure understandable by the computers). In [16] a Context Dimension Tree is used to represent the contest factors with different levels of detail. The model includes constraints and relationships among dimension values to eliminate meaningless context configurations. Such model has been proposed for data tailoring in the database domain, but could be easily adapted to the context of SBAs (see Section 4).

Context-aware applications are able to sense their current context and adapt their behaviour accordingly. Even if automatically enacting adaptation is desired [7], some approaches in the literature suggest the need to have a semi-automatic approach in which users are able order to choose the best adaptation strategy [5]; this would avoid the execution of undesired adaption actions. [9] proposes an approach for context-aware service adaptation based on an ontology for the context modeling and a learning mechanism for the mapping of the context configuration to the appropriate adapted service. Due to the uncertainty behind the context of an application, Cheung et al. [13] propose a fuzzy-based service adaptation model to improve the effectiveness of service adaptation by means of fuzzy theory. The formalization of the service adaptation process is made using fuzzy linguistic variables to define the context situations and the rules for adopting the policies for the implementation of the services.

While the short overview we have provided does not aim at being comprehensive, it should have given the idea that a number of approaches to context modeling and to the development of context-aware adaptable SBAs are being proposed. However, none of them provides a method to develop and execute adaptable SBAs. In [2] we have defined some design guidelines and a life-cycle for the development of adaptable SBAs. In this paper we want to enrich the life-cycle adding information about the context. Therefore, the aim of this paper is to propose a context model for the service based applications and to define how the information contained in the context model could be exploited to enact adaptation during each phase of the life-cycle.

3. ADAPTATION IN SBA

Figure 1 shows the life-cycle for adaptable SBAs we have presented in [2]. It highlights not only the typical design-time iteration cycle that leads to the explicit re-design of an application, but it also introduces a new iteration cycle at run-time that is undertaken in all cases in which the adaptation needs are addressed on-the-fly. The two cycles coexist and support each other during the lifetime of the application. Figure 1 also shows the various adaptation- and monitoring-specific actions (boxes) carried out throughout the life-cycle of the SBA, the main design artifacts that are exploited to perform adaptation (hexagons), and the phases where they are used (dotted lines). At the *requirements engineering and design* phase the adaptation and monitoring requirements are used to perform the design for adaptation and monitoring. During SBA *construction*, together with the construction of the SBA, the corresponding monitors and the adaptation mechanisms are being realized. The *deployment* phase also involves the activities related to adaptation and monitoring: deployment of the adaptation and monitoring mechanisms and deployment time adaptation actions (e.g., binding). During the *operation and management* phase, the run-time monitoring of the relevant properties is executed in order to detect important context and system changes. After this phase the left-side of the life-cycle is executed. Here, we can proceed in two different directions: executing evolution or adaptation of the SBA. In the first case we re-start the right-side of the cycle with the requirements engineering and design phase while in the second case we proceed identifying adaptation needs that can be triggered from monitored events and adaptation requirements. An adaptation need formally characterizes a specific problem-situation that demands for adaptation. It takes into account each monitoring events and tries to answer the following questions: What needs to be adapted? What is the cause? What should be the outcome/what is the aim?. For each adaptation need it is possible to define a set of *suitable strategies* that define the possible ways to achieve the adaptation requirements and needs given the adaptation mechanisms made available. Each adaptation strategy (e.g., service substitution, re-execution, re-negotiation, compensation, etc..[2]) can be characterized by its complexity and its functional and non functional properties. The identification of the most suitable strategy is supported by a *reasoner*, that also bases its decisions on multiple criteria extracted from the current situation and from the knowledge obtained from previous adaptations and executions. After this selection, the *enactment of the adaptation strategy* is performed. The execution of all activities and phases in all runtime phases may be performed autonomously by SBAs or may involve active participation of the various human actors.

4. THE CONTEXT MODEL

The aim of our context model is the formalization of the most relevant aspects characterizing the SBA. The model we propose has been inspired by [16] and is a XML representation of the main components of the context for service based applications. It contains six main dimensions able to describe the status of an application: *Time, Ambient, User, Service, Business* and *Computational* Context. Each dimension in the XML tree, can have sons able to refine each factor. An example of the model is shown in Figure 3 and is discussed in Section 6).

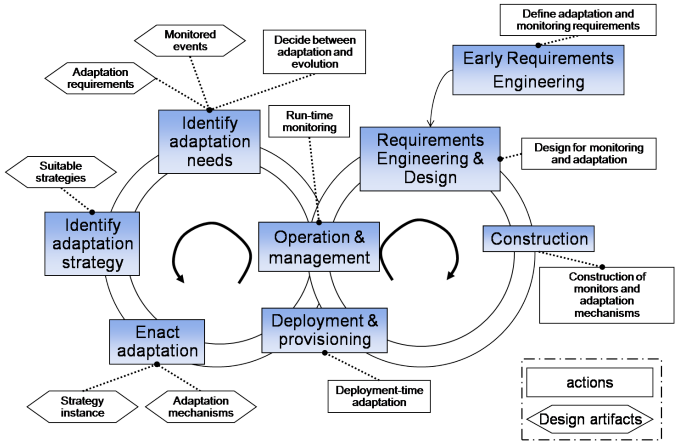


Figure 1: The Life-Cycle of Adaptable SBAs.

The *TimeContext* dimension refers to the information about the time in which the access to the application occurs; it could be expressed in absolute terms (defining a precise date) or it could indicate a part of the day (morning, afternoon, evening, night). The *AmbientContext* can be related to the space factor (expressed in terms of an address by which the user is accessing) or to the environmental condition of the user (the value of some measurable physical parameter). The *UserContext* dimension contains the information about the privileges, the roles, or the preferences the user has in the application. Moreover, such dimension permits to express the goal the user wants to achieve. Information about the services in the application are codified under the *ServiceContext* dimension. This element lists all the services together with their status (if a previous execution reported an error, or if it is available), the time of the last failure (if it makes sense), and the similar services. The latter information could be exploited if a service needs to be substituted with another one. The *BusinessContext* dimension takes into account business application factors. Finally, the *ComputingContext* dimension specifies the software and/or hardware characteristics that are available at the end user side. Such element permits to specify, for example, the device, the operating system, or the web browser the user is using for accessing the application services.

5. CONTEXT-DRIVEN ADAPTATION

The role of context, the influence of its factors on the executability of the applications, and the possible mechanisms for their self-adaptation or human-assisted adaptation, should be properly modelled, designed, and engineered through the whole life-cycle of the system. Referring to the life-cycle of Figure 1, we introduce here a number of actions and design artefacts that are needed to properly build a context model and exploit it to support the adaptation of applications. The context information should be considered and explicitly captured since the very beginning of requirements engineering. In parallel with the elicitation and refinement of requirements we start understanding, which context factors should be considered for the purpose of possible adaptations. The identification and characterization of the context factors of the SBA is continued through the design phase and results in the activity that we call *context modeling*. In parallel, another important aspect of the design activities is the identification of *Context-specific adaptation triggers and requirements*. This refers to the rules and criteria that define the critical context properties and configurations from the adaptation point of view: the context properties that entail SBA adaptation (i.e., to decide *adaptation needs*) and the rules and criteria that define what should be adapted and how (i.e., to decide *adaptation strategies*). These properties will be used to construct the *context-driven adaptation reasoners* that drive the adaptation process at run-time. During the construction phase *contextual monitors* and *contextual adaptation mechanisms* are developed. The first ones are needed to deliver important information about context states and changes during SBA execution. The latter are used to realize different adaptation strategies. Even if some adaptation mechanisms could be independent of a specific context, some others could be dependent on it, for instance, a user-specific service selection policy or the mechanism needed to adapt the result of a service invocation on the basis of the characteristics of the specific devices used by end users (e.g., PC, PDAs, or conventional phones). We remark that the process of identification, modeling, and refinement of the contextual factors and properties is iterative. Indeed, with the definition of new elements of the application and monitoring/adaptation, new factors may be discovered, triggering new adaptation needs and corresponding strategies.

The artifacts and elements developed at design time, i.e., the context model, the contextual monitors, and the adaptation mechanisms are exploited at run-time to activate and drive SBA adaptations. First, the contextual monitors are used to evaluate the *context properties*. Based on them the contextual adaptation reasoners make decisions whether the adaptation should be triggered or not. The decision is driven by the rules identified at design phase and encoded during the construction of the appropriate mechanisms. As a next step, the appropriate adaptation strategy is selected and then enacted by the adaptation framework. As in case of adaptation needs, the selection of the adaptation strategy, as well as its activation, may rely upon and exploit the knowledge about context, e.g., exploit information about the user device to select services and deliver information, about network properties (such as use of GPRS or 3G networks) to activate SLA negotiation.

5.1 Context Modelling

During context modeling the model described in Section 4 is instantiated by identifing the list of context dimensions that can trigger an adaptation or evolution of the functionalities provided by the analyzed SBA. In fact, different context dimensions are relevant for different applications. For example, the *ambient* dimension is irrelevant for all the applications that do not require to change on the basis of the geographical position in which they are used. Once the list of the relevant context dimensions has been defined, the context requirements should be refined by modelling the context dimensions in terms of their reference domain. To enable the design for adaptation aspects, it is first necessary to define if the standard context representation hierarchy is fine for the application or new categories should be defined (e.g., seasons) and new representations identified (e.g., room numbers instead of GPS positions). Second, it is necessary to define the granularity to be used for measuring the context dimen-

sions. For example, as regards our scenario (see Section 6) and the *ambient* dimension, the service for the identification of the health-care public services should adapt its output on the basis of the exact geographical position in which the user accesses it while for the identification of the administrative services, it could be sufficient to retrieve the city from which the user accesses the platform. This first analysis allows the designer to provide the following outcomes:

- The instantiation of the generic context model (Section 4) with respect to the given SBA: the relevant context dimensions, relevant sub-elements of these dimensions, and important set of values (ranges) these elements may have with respect to the analyzed SBA.
- The description of the relationships between SBA elements (services, processes and/or subprocesses, etc.) and the context dimensions in terms of type of impact and specific context values to monitor. This mapping defines the starting point for the definition and modeling of adaptation triggers and adaptation requirements.

We would like to stress the fact that the context modeling is an iterative process: the new dimensions and elements of the model may be added when the application is detailed and new elements (including also monitoring and adaptation mechanisms) are added.

5.2 Adaptation Triggers and Requirements

Once the context model and its relation with the application model is defined, it is necessary to properly capture and define the adaptation aspects. In particular, it is important to define when the contextual changes are critical for the SBA functioning (i.e., adaptation triggers) and what should be done or achieved when these changes take place (adaptation needs). Depending on the context dimension/element and on the specific requirements of an SBA, the definition of adaptation trigger may vary. As a result, the corresponding context monitors needed to detect those trigger will have different, sometimes application-specific, forms and realizations. Furthermore, in certain cases the adaptation trigger does not correspond to some value of a particular single context element, but is characterized by the complex combination of different context dimensions/factors. In the simplest case, such situations may be directly encoded using the capabilities provided by the existing monitoring frameworks, such as Dynamo [1]. In other cases, a sophisticated reasoners may be necessary. In particular, in [6], for instance, a context represents the combination of users personal assets (agenda, location, social relations, and money), and the critical context changes characterize some critical combination of those assets. A dedicated analysis mechanisms is used to identify those situations (asset conflicts) and to trigger adaptation solutions.

Each adaptation trigger can be associated with the adaptation requirements that define what should be done in order to align the SBA with its context. Considering the contextual aspects, it is necessary to take into account that adaptations may be performed (*i*) to *customize* the system in order to fit to the situation in which the application currently operates, (*ii*) to *optimize* the system in order to improve certain (usually quality-of-service) issues and characteristics of the system, or (*iii*) to *prevent and avoid* future faults or undesirable situations in the execution of the system. The customization is often driven by changes in the application context and especially by time, ambient, user, business, and service context dimensions. Optimization is more related to the service and computing context dimensions but can be also triggered by changes in the users' preferences. Finally, prevention is mostly related to the service and computing context since changes of the execution environment might increase the risk of failures and the need for prevention. The adaptation requirements are captured and realized by a set of adaptation strategies. The selection of an adaptation strategy to apply may depend on variety of factors [2], such as scope of the change (i.e., whether the change affects only a single running instance of the SBA or influences the whole model), its impact (the possibility of the application to accomplish its current task or necessity to retract), etc. Besides, the contextual factors should also be considered: the different strategies may be applied upon the same trigger. For example, the decision to substitute a service or to retry with the existing one may depend on the profile of the user. Again, the selection of the adaptation strategy may be characterized by a simple rule or a requires advanced reasoning, based on some ontologies or ad-hoc models, or may involve user decisions [6]. Table 1 relates some of the common adaptation strategies with the changes in different context dimensions. In conclusion, the outcomes of this modeling step include the following components:

- Characterization of adaptation triggers as the description of the context configuration, in which the adaptation should be applied.
- Relation of adaptation triggers to possible adaptation strategies used to realize the corresponding adaptation needs. This relation may be also equipped with the characterization of the contextual conditions that drive the selection of one or another strategy.

5.3 Construction of Contextual Monitors and Adaptation Mechanisms

The results of the analyses performed in the previous phases support the SBA designers in the realization of the monitors and adaptation strategies that should be included in the SBA design. The need for the analysis of the context and the deployment of specific adaptation strategies requires the construction of dedicated platforms and the insertion of monitors able to detect the context changes and trigger the correspondent action. As we already discussed in previous subsection, the realization of such mechanisms may rely on the existing solutions or may require their extension and even completely novel approaches. For example in the Dynamo monitoring framework [1], the monitoring components for context observation are realized as services, which allow for the extension of the platform and even for run-time addition/removal of monitors. Among the other enabling platforms for realizing such context-based monitoring and adaptation solutions one can consider dynamic aspect-oriented programming [11] that provides means for application developers to state when and how behaviour of an application should be adapted. These approaches have

Table 1: Suitability of adaptation strategies to react to context changes

Strategy	Time	Ambient	User	Service	Computing	Business
Service substitution	X	X	X	X	X	X
Re-execution				X	X	
Re-composition				X	X	X
Fail			X	X	X	X
Service concretization	X	X	X	X	X	X
Re-negotiation			X		X	X
Compensation				X	X	
Trigger Evolution		X	X	X	X	X

a limited level of semantic expressiveness since they rely on general-purpose language constructs. Hence, application developers have to express adaptations in the format of the supporting platform which often remains difficult to be understood. Domain-specific languages offer a solution to enable programmers to reason at a higher semantic level. In [10] a language that provides higher-level constructs for expressing the adaptation of application behaviour due to a change in context is proposed. Other approaches are based on middleware-based approaches. For example, in [3], authors present a framework for self-adaptive components that follows the Event-Condition-Action pattern and makes use of events emitted by a context-aware service that provides information about the execution context of the application. Furthermore, various rule engines can be also used. In [8], a dynamic adaptation framework that adapts service objects in a context-aware policy-based manner, using metatypes is proposed. The system includes in the policy document a-priori information for adaptation in the form of default behaviours and known adaptations. The system is also able to deal with adaptation requirements that were unprecedented when the service was designed and compiled. In fact, reconfiguration intelligence can be incorporated at runtime by modifying the policy declarations and the inclusion of new behaviours in the form of new metatypes. The outcomes of this construction step include the following components:

- Design and realization of monitors able to detect changes in the context dimensions on the basis of the specifications defined in the previous phase.
- Realization of a platform that on the basis of the context analysis define and trigger the corresponding adaptation action.

6. PROCESS EVALUATION

In this section we want to explain how the context-aware design process described in the section 5 is applied to a case study in the e-government domain that has been defined in the S-Cube Project [12]. The idea is to build an adaptable SBA that realizes e-government processes able to support citizens in various activities. The application aims at provisioning and integrating various public services, including the health-care services, administrative procedures towards citizens, information services, personal assistance services, etc. These services range from Internet services (e.g., for booking a visit to a doctor online, online payment, route planning) to local services (e.g., specific medical centers and labs, monetary assistance), to personal and human-provided ones (medical assistance at home, fiscal procedures, monetary assistance, census operations). Moreover, public authorities own and expose to the citizens various e-government processes and procedures, where the single services of private bodies and public organizations are composed. An example of an integrated process is the reservation of a medical visit, where the reservation service is integrated with the local services of the medical centers, with the route planning system and public transportation services to drive the patient to the doctor and inform it about the available transport means. Figure 2 presents the general picture of the scenario, it includes all possible services, subdivided in different categories, and different actors of the application: Citizen, Service Providers, Public Authorities and the Service Integrator. The Service Integrator is the principal actor that realizes the e-government SBA using the principles that we have defined in this paper.

In the scenario introduced above different contextual dimensions are heavily exploited, ranging from computational context (e.g., devices and channels used by the citizens to access/receive information from the administrative services), to physical context (e.g, user location), user context (its. preferences, social status, medical card), and even to business aspects (e.g., the emergency mode of the e-government procedures). In particular, let us consider the Health-care public service that one user can access to search for a doctor. If the request occurs during the night, the list of the available doctors is different with respect to the one returned during the morning. For this reason, the *TimeContext* dimension will be expressed by means of the *PartOfDay* element whose values will belong to the following set: {*morning, afternoon, evening, night*} (see the figure 3). In a similar way, some of the service outputs could depend on the location of the user accessing the system. In fact, if a citizen wants the list of the nearest pharmacies, the sorting of the returned items depends on the zipcode of the area from which the request comes. In a similar way the behaviour of the application will depend on the situation (normal life or emergency conditions) the users are living. The study of the application and of the involved services will guide the definition of the set of values associated to the context dimensions. An example of a simple context model for the proposed scenario can be found in the Figure 3; the model reports, for each considered dimension, all the admissible values.

After the definition of the context model, it is possible to go through the next phase devoted to the identification of the associations of the critical context changes (adaptation triggers) to the adaptation needs. In Table 2, for each service involved in the application the relationship with the context factors is highlighted. It is possible to notice, for example, that the access to the Administrative or to the Health-care public services, could depend on the values of the *TimeContext* dimension in the context model.

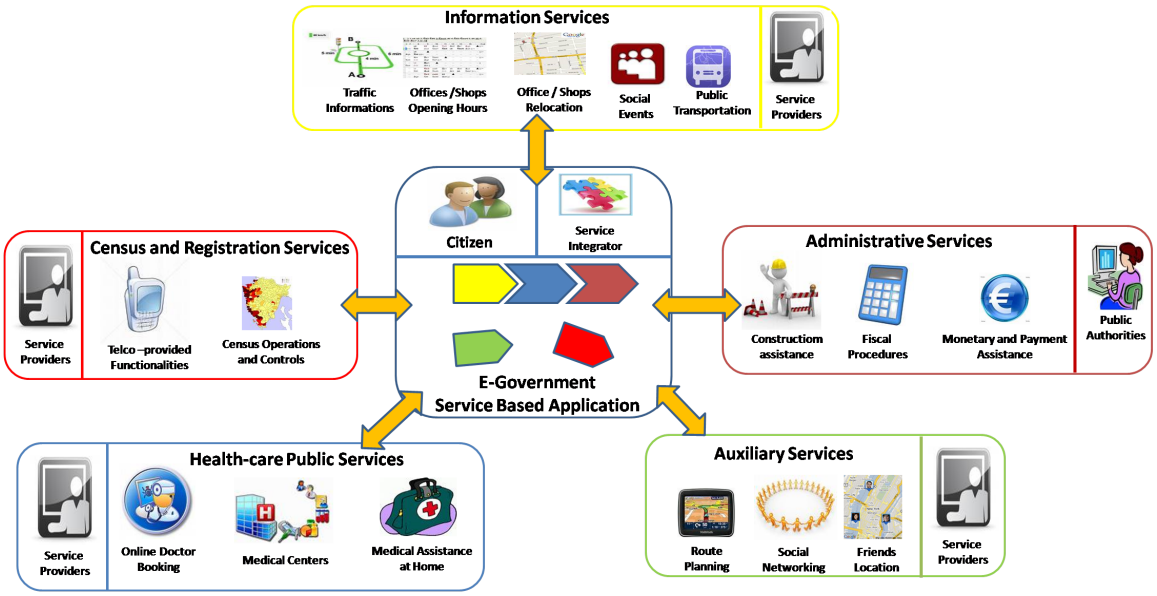


Figure 2: General picture of the e-government scenario.

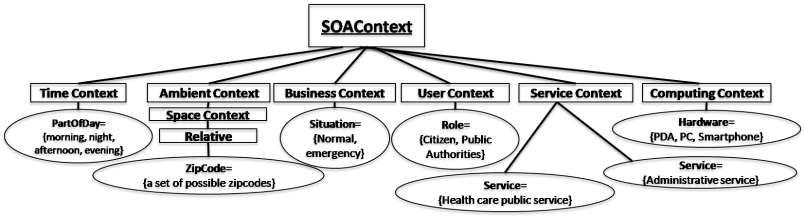


Figure 3: The Context Model for the e-Government scenario.

Table 2: Service/Context table for the E-government scenario

Application	Time	Ambient	User	Service	Computing	Business
Health-care public services	X	X	X	X	X	X
Administrative services	X	X	X	X	X	X
Census and registration services		X	X	X	X	X
Information services	X	X	X	X	X	X
Auxiliary services		X		X	X	X

Table 3: Service/Adaptation Strategy table for the e-government scenario

Adaptation Strategy	Health-care public services	Administrative services	Census and registration services	Information services	Auxiliary services
Service substitution	X			X	X
Re-execution	X	X	X	X	X
Re-composition	X	X	X	X	X
Fail	X	X	X	X	X
Service concretization	X			X	X
Re-negotiation	X			X	X
Compensation	X	X	X		X
Trigger Evolution	X	X	X		

Changes in the context have to be managed and suitable adaptation strategies have to be defined. Referring to the e-government scenario, if we consider a mobile user accessing to the e-Health service, we can image that modifications in the location (expressed using the zipcode for the specific context dimension) will require the re-execution or the substitution of the specific e-Health service. Moreover the transition from a normal to emergency situation could require the recomposition of the application. During the second phase of the process, all the context changes and the corresponding adaptation strategies have to be identified. On the basis of the analyzed scenario, the context requirements, and the characteristics of the adaptation strategies, process designers in the last phase should build a table *Ser-*

vice/Adaptation Strategy in which for each service the selected adaptation strategies are identified. Table 3 details the *Service/Adaptation Strategy* table for the e-government scenario. Since almost all the functionalities depend on the whole set of the context dimensions, they are also suitable for the adoption of a large set of adaptation strategies. The public ownership of the administrative and census and registration services and thus the unavailability of a registry containing similar services make instead the service substitution and concretization not adoptable. Compensation is not suitable for all the Information services since the only read interaction with these services does not require the intervention of compensation operations. The table *Service/Adaptation Strategy* provides a comprehensive view of all the applicable strategies and thus supports the designer in the identification of the elements to design and develop to enable the construction of a contextual adaptive SBA. For each service and for each related context dimension, one or more adaptation strategies should be selected on the basis of technical constraints and functional and non-functional requirements. For example, for all the critical services in which a rapid response is needed, the adoption of adaptation strategies that increase the response time (e.g., re-execution, re-negotiation, re-composition, trigger evolution) is not acceptable.

7. CONCLUSION AND FUTURE WORK

This paper focuses on the role of the context in the adaptation activities. It proposes a framework to support the design of SBAs that targets the adaptation requirements raised by context changes. The approach has been described on the basis of a novel life-cycle that emphasizes the relevance of the context elements in the different facets of adaptation, both during the design phase and at run-time. The paper considers all the issues related to the design of SBAs able to evolve together with the requirements and the execution context. The context has been modeled by considering a set of all the dimensions that can generally influence the system behaviour. On the basis of this context model, the proposed approach provides guidelines for the identification of the relevant context dimensions to monitor and for the definition of the adaptation triggers able to link context changes with suitable adaptation strategies. The effectiveness of the discussed principles and guidelines has been evaluated by considering a real case study based on an e-governement scenario. Results witness the capability of the context-driven adaptation process to capture the key aspects of adaptation and support designers from the requirements elicitation to the construction of proper adaptation mechanisms. Our future roadmap includes a refinement of the adaptation process presented in this paper, its formalization, and validation. We also intend to work on the development of mechanisms and tools supporting the methodology, building on top of the actions and artifacts identified in the proposed life-cycle.

8. REFERENCES

[1] L. Baresi and S. Guinea. Dynamo: Dynamic monitoring of ws-bpel processes. In *ICSOC*, pages 478–483, 2005.

[2] A. Bucchiarone, C. Cappiello, E. di Nitto, R. Kazhamiakin, V. Mazza, and M. Pistore. Design for adaptation of Service-Based applications: Main issues and requirements. In *International Workshop on Engineering Service-Oriented Applications (to appear)*, 2009.

[3] P.-C. David and T. Ledoux. Wildcat: a generic framework for context-aware applications. In *MPAC*, pages 1–7, 2005.

[4] A. K. Dey and G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness. *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, 2000.

[5] C. Efstratiou, K. Cheverst, N. Davies, and A. Friday. An architecture for the effective support of adaptive context-aware applications. In *MDM '01: Proceedings of the Second International Conference on Mobile Data Management*, pages 15–26, London, UK, 2001. Springer-Verlag.

[6] R. Kazhamiakin, P. Bertoli, M. Paolucci, M. Pistore, and M. Wagner. Having Services "YourWay!": Towards User-Centric Composition of Mobile Services. In *Future Internet Symposium*, 2008.

[7] J. Keeney and V. Cahill. Chisel: A policy-driven, context-aware, dynamic adaptation framework. *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:3, 2003.

[8] J. Keeney and V. Cahill. Chisel: A policy-driven, context-aware, dynamic adaptation framework. In *POLICY*, pages 3–14, 2003.

[9] M. Moez, C. Tadj, and C. ben Amar. Context modeling and context-aware service adaptation for pervasive computing systems. *International Journal of Computer and Information Science and Engineering*, 2008.

[10] J. Munnelly, S. Fritsch, and S. Clarke. An aspect-oriented approach to the modularisation of context. In *PerCom*, pages 114–124, 2007.

[11] A. Nicoara and G. Alonso. Dynamic aop with prose. In *International Workshop on Adaptive and Self-Managing Enterprise Applications (ASMEA)*, pages 125–138, 2005.

[12] E. D. Nitto, V. Mazza, and A. Mocci. Collection of industrial best practices, scenarios and business cases, 2009.

[13] J. C. Ronnie Cheung, Gang Yao and A. Chan. A fuzzy service adaptation engine for context-aware mobile computing middleware. *International Journal of Pervasive Computing and Communications*, 2008.

[14] S-Cube. Codified human-computer interaction (HCI) knowledge and context factors, 2009.

[15] T. Strang and C. L. Popien. A context modeling survey, September 2004.

[16] L. Tanca, A. Miele, and E. Quintarelli. A methodology for preference-based personalization of contextual data. In *ACM EDBT 2009*, 2009.

Modelling and Automated Composition of User-Centric Services

Raman Kazhamiakin[1], Massimo Paolucci[2], Marco Pistore[1] and Heorhi Raik[1]

[1] Fondazione Bruno Kessler, via Sommarive 18, Trento TN 38050, Italy
[raman,pistore,raik]@fbk.eu
[2] DoCoMo Euro-Labs, Landsberger Strasse 312, 80687 Munich, Germany
paolucci@docomolab-euro.com

Abstract. User-centric services bring additional constraints to the problem of automated service composition. While in business-centric settings the services are orchestrated in order to accomplish a specific business task, user-centric service composition should allow the user to decide and control which tasks are executed and how. This requires the ability not only to automatically compose different, often unrelated, services on the fly, but also to generate a flexible interaction protocol that allows the user to control and coordinate composition execution. In this paper we present a novel automated composition approach that aims to support user-centric service provisioning. Specifically, we associate the service to so-called service objects and provide a declarative notation to express composition requirements in terms of the evolution of those objects. On top of these objects we also define the user control activities and constraints. Using the automated planning techniques, our approach generates a service composition that orchestrates services in a way it is requested by the user.

1 Introduction

In the past decade, the advances in technology allowed numerous service providers to introduce thousands of new electronic services as well as to create service-oriented adapters for conventional services. In these settings, the ability to efficiently integrate and compose those services in order to obtain new functionalities becomes one of the key factors for the wide adoption of the service-oriented paradigm. A variety of technologies and approaches is already available to facilitate the development of service compositions.

With the growth of the service market more and more prominent role is gained by the *user-centric services*, i.e., the services that are intended to be consumed directly by the user (e.g., personal agenda, electronic maps, on-line flight booking and check-in, restaurant finder, etc.), as opposed to business-centric (or B2B) services, which are used to provide business-to-business cooperation. Although the rapid emergence of user-centric services is the trend of the last few years, some works already distinguished them as a separate group of services [5], and some even considered their features from the service composition perspective ([6] and [9]). While from the technological viewpoint user-centric composition may rely on similar composition solutions and standards, the way such services are composed and delivered to the users should adhere to some specific yet significant requirements and constraints. In particular:

– B2B service composition aims at realizing a specific business task in a structured way (e.g., by implementing a corresponding business process). In the user-centric settings the goal is to continuously support the user in performing variety of different tasks, being able to react to the changes in her plans and decisions and to propagate those changes to the composed services. This requires different ways to capture composition requirements, shifting from the definition of an ultimate composition goal to the definition of the rules and constraints on continuous service coordination.
– The execution of the business-centric composition is driven and controlled by the rationale defined by the business goal behind that composition. The execution of the composed user-centric services is driven and controlled by the user deciding which activity to execute, when it should be executed and how. The key issue here is to identify appropriate interaction means that would deliver controllability and awareness of the execution to the user in an intuitive way.
– In both cases the composition is being constructed from the service models and composition requirements defined by the designer at design time. In the user-centric settings, however, the decision of which services should be integrated and composed is left to the user, which makes the service composition a purely run-time activity. First, this makes the use of automated service composition techniques unavoidable. Second, there is a need for the appropriate modelling techniques, where the composition requirements are sufficiently abstracted from the service implementations. Third, this makes the problem of controlling the execution of the composition by the user even more complex, since the corresponding interaction protocols should be constructed and provided to the user at run-time, in a composition-specific manner.

In this paper we present a novel service composition framework which aims to address these challenges. Building upon our previous techniques for automated service composition [4], this approach provides the capabilities specifically targeting the needs of user-centric service compositions. In this framework, we propose:

– A simple yet expressive language for control flow composition requirements, being able to deal with the problems specific to user-centric service compositions just discussed. The language relies on the service model, where the essential service properties are abstracted from the low-level service implementations, thus enabling dynamic re-use of those models at run-time.
– A set of principles capturing the patterns of interaction with the user, so that he is always in control and aware of the execution of his services in an intuitive way.
– An automated support for the service composition and derivation of the corresponding user protocol, through which the user can trigger different activities, control their execution, and receive relevant information about the execution flow.

The rest of the paper is structured in the following way. In Section 2 we present out motivating example and discuss the main challenges we face. In Section 3, the general approach is outlined. Section 4 contains some background on wired planning. In Section 5, the formalization of the composition model is given. Section 6 is devoted to the implementation of the composition framework and experiments. In Section 7 we discuss related work.

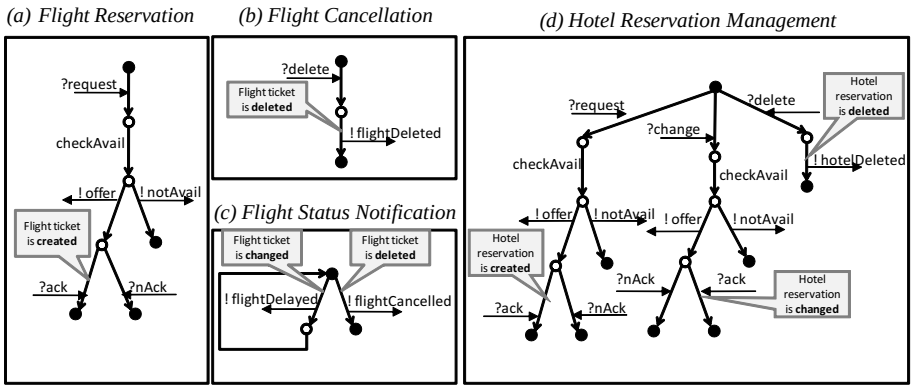


Fig. 1. Protocols of components in travel domain scenario

2 Motivating Example

For the illustration purposes, we use a case study from the travel domain, which is widely used in the literature on service composition. In this case study, the user deals with services that manage different activities over flight bookings and hotel reservations. More precisely, these services are (Fig. 1):

1. *A flight booking service (a)*. Upon the user request, it sends an offer and waits for the confirmation.
2. *A flight cancellation service (b)*. It simply cancels the flight upon the user request.
3. *A flight status notification service (c)*. It notifies the user about flight delays and cancellations.
4. *A hotel reservation management service (d)*. This is a complex service that provides means to create, modify and cancel hotel reservations.

In this scenario our goal is not simply to allow the user to book a flight and a hotel for the same trip, but to continuously coordinate their evolution when the changes are required by the user (e.g., the user wants to cancel the flight) or triggered by exogenous events (e.g., flight is delayed). Specifically, the coordination requirements may be:

– a flight ticket and hotel reservation should be booked together (i.e., transactionally).
– when the flight is delayed, the hotel reservation should be postponed, and if this is not possible, both should be cancelled.
– when the flight is cancelled, the hotel reservation should be cancelled too.

Furthermore, the composition should enable the user to control the execution. That is, the user should be able to change his objects (e.g., to cancel the flight or to modify the hotel reservation), to be aware of the changes (e.g., to receive notifications), and to make important decisions (e.g., to accept or reject proposed hotel options).

Finally, we expect that the specific service implementations (i.e., specific flight and hotel booking services) are selected dynamically, and the concrete composition is created at run-time.

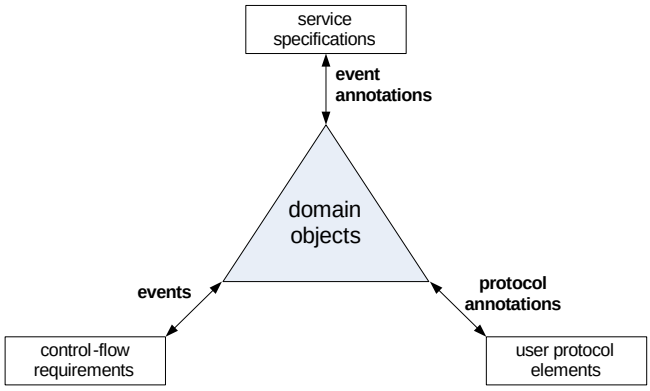
The requirements posed by the presented scenario make the service composition problem much more challenging than in the similar scenarios used to evaluate existing composition approaches. In particular, the following problems should be addressed:

- *Heterogeneity of service implementations*. There may be no single service that can completely manage an entity (e.g., a flight ticket is managed by three different services, some of them may even belong to different service providers). Moreover, different provider may define the same service in different ways.
- *Continuous service coordination*. There is a need to continuously coordinate a few entities (a flight ticket and a hotel reservation) rather than simply perform a single task.
- *Re-use of composition requirements*. The complexity of control-flow requirements and dynamicity of composition instantiation raises the question of reusing them with different service implementations. In our example, once specified, composition requirements should be applicable to different implementations of flight and hotel management services.
- *Dynamicity of user protocols*. While there is clearly a need to enable the user to manipulate services and to control the composition execution, the way the user interacts with the composition, i.e., user protocol, strictly depends on the participating services and on their implementations. Such a protocol can hardly be predefined for all possible realizations.

3 Overview

In order to address the problems outlined in Section 2, we introduce a novel service modelling and composition approach. In this section we present the key elements of the approach, namely, the composition model and the composition construction process.

Composition Model. Our composition model is shown in the following figure:



The central element of this model is the definition of *domain objects* used to represent the entities participating to the composition (e.g., a flight ticket, or a hotel reservation) and their evolution at a high level of abstraction. Details of concrete service implementations are captured in the definition of *service specifications*. These specifications are related to domain objects through event annotations that define how the execution of services makes the corresponding object evolve. The specification of the *control-flow requirements* over the service composition is given in terms of evolution of domain objects only, regardless of the specific services realizing those objects. Similarly, the *user protocol elements* capturing the requirements on the user interactions with the composition are related to the composed objects, and not to the service implementations nor to composition requirements. We remark that with this distinction we are able to address the problem of service implementation heterogeneity, re-use of composition requirements in different settings, and automatically come up with the necessary user protocol regardless of specific composition scenario and the services participating in it.

The domain objects used in our model are represented with an *object diagram*, a state diagram that defines possible object states and transitions between them (e.g., a flight ticket can be booked, then paid, then checked-in or cancelled or delayed etc.). In fact, an object diagram is the representation of an object life-cycle, where transitions correspond to the activities that can be performed over the object (e.g., flight cancellation) or external event that can happen to the object (e.g., flight delay committed by the airline).

Following the real-world phenomena, in our model we allow for stateful services with complex protocol, asynchronous and non-deterministic behavior. These services may be defined using standard notations (e.g., BPEL). To link service specifications and domain objects, we annotate some of the actions in service protocols with the events of object evolutions. As such, through domain objects our model is able to capture complex and diverse service implementations and to relate independent services managing the same entity.

The language for the specification of control-flow requirements is defined on top of object diagrams, i.e., in terms of states and events. The language aims to express various coordination goals, i.e., alignment of states of different domain objects, reaction to various events, and so on. With this language our approach supports continuous coordination requirements, enabling their reuse since the requirements are detached from service implementations.

When the composition instance and constituent services are defined, our approach automatically derives a specific user protocol for the composition. Doing this, the following requirements are considered:

- The user should have a way to manipulate the objects using available services.
- The user should be able to receive notifications about object evolution.
- During the execution, critical decisions should be delegated to the user.

Following these requirements, the corresponding protocol elements are defined in our model. Specifically, these elements are used to enable and control the activation of certain object transitions by the user, to inform the user about the changes, and to capture critical decisions and the interaction with the user on those decisions. Formal definition

of the elements is given in Section 5.4.

Composition Process. The composition process consists of two major phases: design time and run time. At *design time*, IT experts model the domain objects and associate the existing services to those objects through service annotations. After that, it is possible to define groups of domain objects that are likely to be composed with each other (e.g., a flight ticket, a hotel reservation and a car rental are likely to be coordinated in the context of trip planning). For each such group of objects, a set of control-flow requirements is specified which defines how to coordinate them. In fact, these requirements define a sort of *composition template* for coordinating certain objects (e.g., the template on how to manage a trip consisting of a flight, a hotel and a car rental). Finally, the resulting specifications, namely domain objects, services, and composition templates, are stored in some service repositories. The way the repositories are defined, accessed, and managed is outside of the scope of this paper.

At *run time*, the template is instantiated with concrete objects, and services that can manage these objects are offered to the user. Once the service implementations are chosen, we use planning techniques to automatically build a composite service. Our planning-based composition tool takes as input the models of object diagrams, annotated service specifications, and control-flow requirements, we translate these models into a planning domain with planning goals over this domain. The plan obtained is intended to implement the behaviour of the composite service that meets the composition requirements specified. Finally, it is translated back to one of the standard composition languages (e.g., BPEL).

We remark that the ability to select service implementations at run time is a very important capability that we consider. The fact that implementations of the same services can be radically different (e.g., implementations of a flight booking service provided by various airlines) makes the realization of such capability far from trivial and indeed requires advanced planning techniques available at run time.

4 Background on Service Composition via Planning

In our approach to service composition we rely on the composition framework presented in [10] (planning in asynchronous domains). In that research, a planning domain is derived from service specifications, composition requirements are formalized as a planning goal, and advanced planning algorithms are applied to the planning problem to generate the composite service. The advantages of the approach are an asynchronous communication model, the ability to deal with stateful and non-deterministic services, considering preference-based (reachability) requirements on services. This is why it provides a good basis to build upon. Here we discuss the main concepts of that formal framework.

A planning domain is formally defined as a state transition system, i.e., a dynamic system that can be in one of its *states* (some of which are *initial states* and/or *accepting states*) and can evolve to new states as a result of performing some *actions*. Actions can be *input* actions, which represent the reception of messages, *output* actions, which

represent messages sent to external services, and internal action τ, modelling internal computations and decisions.

Definition 1 (STS). *A state transition system is a tuple $\langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{S}^F, \mathcal{F} \rangle$, where:*

- *$\mathcal{S}$ is the set of states and $\mathcal{S}^0 \subseteq \mathcal{S}$ are the initial states;*
- *$\mathcal{I}$ and $\mathcal{O}$ are the input and output actions respectively;*
- *$\mathcal{R} \subseteq \mathcal{S} \times Bool \times (\mathcal{I} \cup \mathcal{O} \cup \{\tau\}) \times \mathcal{S}$ is a transition relation, where $Bool$ represents all boolean expressions over propositions $\mathcal{P}$;*
- *$\mathcal{S}^F \subseteq \mathcal{S}$ is the set of accepting states;*
- *$\mathcal{F} : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ is the labelling function.*

Given a labeling function the $\mathcal{F} : \mathcal{S} \rightarrow 2^{\mathcal{P}}$, we can define whether a boolean expression $b \in Bool$ over propositions in $\mathcal{P}$ holds in state $s \in \mathcal{S}$, written $s, \mathcal{F} \models b$. The definition is the classical one for validity of boolean expressions.

The transitions of STS are guarded: a transition $\langle s, b, a, s' \rangle$ is possible in the state s only if the guard expression b holds in that state, i.e., $s, \mathcal{F} \models b$. A run π of STS is a finite sequence of transitions $\pi = \langle s_0, b_0, a_0, s_1 \rangle, \ldots, \langle s_n, b_n, a_n, s_{n+1} \rangle$, with $a_i \in \mathcal{I} \cup \mathcal{O} \cup \{\tau\}$, $s_i, \mathcal{F} \models b_i$, $s_0 \in \mathcal{S}^0$, and $s_{n+1} \in \mathcal{S}^F$. The set of all runs of a STS Σ is denoted with $\Pi(\Sigma)$.

Component services can be recast as STSs, and, given a set of component services $W_1, \ldots, W_n$, the planning domain Σ is defined as a synchronous product of the all the STSs of the component services: $\Sigma = \Sigma_{W_1} \| \ldots \| \Sigma_{W_n}$. The synchronous product $\Sigma_1 \| \Sigma_2$ models the fact that the systems Σ_1 and Σ_2 evolve simultaneously on common actions and independently on actions belonging to a single system. A composed service can also be represented as a state transition system Σ_c. Its aim is to control the planning domain defined by the component services. The interactions of Σ_c and Σ are modelled by the following notion of a controlled system.

Definition 2 (Controlled System).
Let $\Sigma = \langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{S}^F, \mathcal{F} \rangle$ and $\Sigma_c = \langle \mathcal{S}_c, \mathcal{S}_c^0, \mathcal{I}, \mathcal{O}, \mathcal{R}_c, \mathcal{S}_c^F, \mathcal{F}_c \rangle$ be two state transition systems. The state transition system $\Sigma_c \triangleright \Sigma$, describing the behaviors of system Σ when controlled by Σ_c, is defined as follows:

$$\Sigma_c \triangleright \Sigma = \langle \mathcal{S}_c \times \mathcal{S}, \mathcal{S}_c^0 \times \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}_c \triangleright \mathcal{R}, \mathcal{S}_c^F \times \mathcal{S}^F, \mathcal{F}_c \cup \mathcal{F} \rangle$$

where:

$$\langle (s_c, s), (b_c \wedge b), a, (s_c', s') \rangle \in (\mathcal{R}_c \triangleright \mathcal{R}), \text{ if}$$
$$\langle s_c, b_c, a, s_c' \rangle \in \mathcal{R}_c \text{ and } \langle s, b, a, s' \rangle \in \mathcal{R}.$$

In this setting, generating a composed service Σ_c that guarantees the satisfaction of a composition goal ρ is formalized by requiring that the controlled system $\Sigma_c \triangleright \Sigma$ satisfies the requirement ρ: $\Sigma_c \triangleright \Sigma \models \rho$. In [11] it is shown how planning for preference-ordered goals may be applied for this purpose, considering a list $\rho = (g_1, g_2, \ldots, g_n)$ of alternative requirements where each g_i is a reachability goal.

In our work, we will use this idea as a basis on top of which we will build our composition model.

5 Formal Model

In this section, we formally describe each part of our composition framework introduced in section 3. The model is an extension of the formalization presented in [4]. This extension, on the one hand, enriches the modelling capabilities of the approach and, on the other hand, allows for capturing specific properties of user-centric service composition.

5.1 Domain Objects

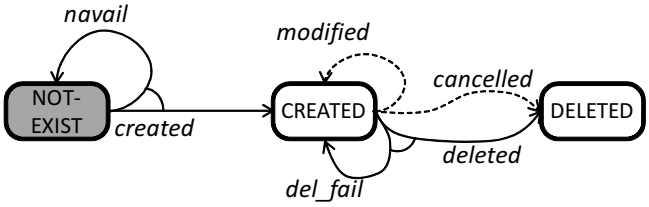
A domain object is intended to model some entity that we can manage using services. One service composition is likely to manage more than one domain object (in the motivating example, we manage two domain objects: a flight ticket and a hotel reservation). Every domain object is modelled with an object diagram, which is a state transition system. Transitions in an object diagram stand for activities that can be performed over the object (e.g., to book a flight ticket). To address object diagram transitions in control-flow requirements and service annotations, we label transitions with object events. Differently from [4], in this work we distinguish between two types of transitions. Transitions that are triggered by executing a certain service are called *controllable* (e.g., hotel booking) because the user can decide whether to execute them and when. These type of transitions can have more than one final state corresponding to different non-deterministic outcomes of service execution. Each non-deterministic outcome is labelled with its own event. Transitions corresponding to external notifications (e.g., flight delay notification) are called *uncontrollable* because they can happen at any time and are out of the user's control. These transitions are deterministic and have exactly one final state.

Definition 3 (Object Diagram). *An* object diagram *representing object O is a tuple $\langle L, L_0, \mathcal{E}, T \rangle$, where:*

- *L is a finite set of object configurations and $L_0 \subseteq L$ is a set of initial configurations;*
- *$\mathcal{E}$ is a set of possible events that reflect the evolution of the object;*
- *$T \subseteq L \times (\mathcal{E} \times L)^+$ is a transition relation that defines non-deterministic evolution of an object, based on events. We distinguish between controllable transitions T_c and uncontrollable transitions T_u such that $T = T_c \cup T_u$ and $T_c \cap T_u = \emptyset$.*

We remark that transition relation in domain objects allows for a few effects per transition, each leading to its own final state. Such a model takes into account that each activity over object may have a few possible non-deterministic outcomes (see the example below).

Example 1 (Flight Ticket). A possible object diagram for domain object Flight Ticket may be represented as follows:



In this diagram, solid lines correspond to controllable and dashed lines to uncontrollable transitions. Non-deterministic outcomes of the same transition are marked with an arc. The diagram has three different configurations *not-exist*, *created* and *deleted*. The transition corresponding to flight booking is an example of a controllable transition with non-deterministic outcomes. The object moves to state *created* when the flight booking service is executed successfully (event "created"). At the same time, non-deterministic fail of this service is still possible (e.g., event "navail" represents non-availability of the flight with requested attributes). Formally, this transition can be represented as $T_{cr} = (\textit{not-exist}, \{(created, \textit{created}), (navail, \textit{not-exist})\})$. In state *created*, two uncontrollable transitions are presented: if the flight is delayed (event "modified") or cancelled (event "cancelled") by the airline. The ticket cancellation can also be performed in a controllable way using a corresponding service (events "deleted" and "del_fail").

5.2 Services

In this research, we reuse the model of annotated services presented in [4]. We deal with stateful services represented with a service protocol (e.g., a BPEL process) and a stateless interface (e.g., a WSDL specification). Every service is related to a specific domain object through special annotations (in certain sense, we use a service to produce some effect on a corresponding object). These annotations appear within the activities of the service protocol: an activity may be annotated with a set of events pertaining to the corresponding object. This implicitly defines how the execution of the service changes the object.

We use *annotated state transition system* (ASTS) as a model of service. "Annotated" means that each transition in such an STS may be labelled with object diagram events. The execution of a labelled transition propagates labelling events to the object. If, for example, a service transition is annotated with event $to^{\mathsf{e}}(l, o)$, then object o moves to configuration l when this transition takes place.

Definition 4 (Annotated State Transition System). *An annotated STS Σ is a tuple $\langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{E}, \mathcal{R} \rangle$ where:*

- *$\mathcal{E}$ is the set of events;*
- *$\mathcal{R} \subseteq \mathcal{S} \times (\mathcal{I} \cup \mathcal{O} \cup \{\tau\}) \times \mathcal{E}^* \times \mathcal{S}$ is the transition relation.*

The semantics of the annotated transition is intuitively described as follows. Let object o have a set of events $\mathcal{E}$, and a service transition $(s, a, \varepsilon, s') \in \mathcal{R}$, where $s, s' \in \mathcal{S}$, $a \in (\mathcal{I} \cup \mathcal{O} \cup \{\tau\})$, and $\varepsilon \subseteq \mathcal{E}$.

The transition is considered to be *applicable* to the object o if the object is in some configuration l, and either $\varepsilon = \emptyset$, or there exists an object transition (l, ε', l'), such that $\varepsilon' \subseteq \varepsilon$. Once this transition is performed, in the first case the object will remain in the same configuration, while in the second case it will evolve to the configuration l'.

Example 2. An annotated STS of the Flight Cancellation Service (Fig. 1, b) is defined as $\Sigma = \langle \{s_0, s_1, s_2\}, s_0 \}, \{delete\}, \{flightDeleted\}, \mathcal{E}, \mathcal{R} \rangle$. The output operation "flightDeleted" is annotated with the event $d^{\mathsf{e}}(f)$ (flight is "deleted"), as the

service provider confirms the flight ticket cancellation. That is, $\mathcal{E} = \{d^{\mathsf{e}}(f)\}$, and $\mathcal{R} = \{(l_0, ?delete, \emptyset, l_1), (l_1, !flightDeleted, \{d^{\mathsf{e}}(f)\}, l_2)\}$.

We remark that service protocols can feature branching. For example, in the flight booking service (Fig. 1(a)) after the flight options are offered, it is possible either to accept or to reject the offer. For such protocols, more than one execution path is available. At any branching point, the decision on which branch to take is delegated to the user and is a part of the derived user protocol (for details see subsection 5.4).

5.3 Control Flow Requirements

To express control-flow requirements, we use an easy-to-specify language that is independent from service implementations and expressive enough to specify complex requirements and can take into account the user's behaviour. For those reasons, this language (i) expresses requirements on the evolution of objects, synchronizing object diagrams and defining "stable" configurations for the system of object; (ii) specifies requirements as reaction rules, which define how the composite service shall react to object events in different situations. Differently from [4], we introduce a "try-catch" construction. The intuition is to handle the situation where we want some effect to be provided, but expect that while executing services some fails may happen and these fails will require additional reaction.

Definition 5 (Composition Constraint). *A composition requirement is defined with the following generic template*

$$premise \Longrightarrow (reaction_1 \succ \ \ldots \ \succ reaction_n),$$

where $premise \equiv s^{\mathsf{s}}(o) \mid e^{\mathsf{e}}(o) \mid pr_1 \vee pr_2 \mid pr_1 \wedge pr_2$

and $reaction \equiv s^{\mathsf{s}}(o) \mid e^{\mathsf{e}}(o) \mid r_1 \vee r_2 \mid r_1 \wedge r_2 \mid \ try(r)$
$catch(pr_1) : r_1$
$\ldots$
$catch(pr_n) : r_n$

Here pr_1, pr_2 *and* pr_n *are premises,* r, r_1, r_2 *and* r_n *are reactions,* $s^{\mathsf{s}}(o)$ *is used to define the fact that the object* o *is in the configuration* s*, and* $e^{\mathsf{e}}(o)$ *defines that the event* e *of the object* o *has taken place,* $try(r)catch(pr_i) : r_i$ *construction means that it is necessary to do all possible to provide reaction* r*, but if, while trying to provide this reaction, situation* pr_i *(premise of the catch block) has happened,* r_i *should be provided instead.*

The left side of the constraint defines the "premise" and the right side defines a set of "reactions" that we expect from the composite service (reactions are ordered according to their priorities). More precisely, the constraint states that if the situation described in the "premise" has happened, one of the situation described in the corresponding "reactions" has to be provided with respect to given priorities. Expression $try(r)$ has to be interpreted as "do your best to provide r, bur even if you finally failed you are successfully done with $try(r)$".

Example 3. The composition requirements of the motivating example in Section 2 can be written down using our language in the following way:

1. $cr^{\mathsf{s}}(h) \vee cr^{\mathsf{s}}(f) \Longrightarrow (cr^{\mathsf{s}}(f) \wedge cr^{\mathsf{s}}(h))$

   Here, the transactional creation of two objects h (hotel reservation) and f (flight ticket) is defined, i. e., mandatory bring both objects to the state "cr" (created) in case at least one of them is created. In such way we define "stable" configuration of a system of objects (both are created).
2. $mod^{\mathsf{e}}(f) \Longrightarrow try(mod^{\mathsf{e}}(h))$
   $catch(refused^{\mathsf{e}}(h) \vee mod\_navail^{\mathsf{e}}(h) :$
   $try(del^{\mathsf{s}}(h)) \wedge try(del^{\mathsf{s}}(f))$

   This constraint states that once the flight is delayed (event $mod^{\mathsf{e}}(f)$), the composition has to do its best to modify the hotel reservation (i. e., to trigger event $mod^{\mathsf{e}}(h)$ for the hotel reservation). If specific types of fails happen while changing the hotel reservation (non-availability $mod\_navail^{\mathsf{e}}(h)$ or the user's refusal of the new hotel offer $refused^{\mathsf{e}}(h)$) then the composition has to try to delete both objects. This is an example of requirements that specify reaction to events that the composite service has to provide.
3. $d^{\mathsf{e}}(f) \Longrightarrow (try(d^{\mathsf{e}}(h))$.

   If the flight is cancelled (event $d^{\mathsf{e}}(f)$), the composition tries to cancel the hotel reservation (event $d^{\mathsf{e}}(h)$).

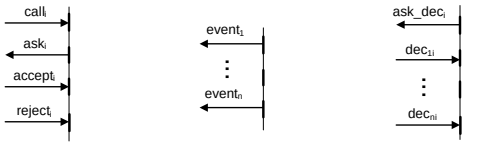
The resulting composite process is supposed to continuously satisfy all the control-flow requirements specified and thus consistently manage the whole life cycle of two object.

We want to emphasize that, due to the fact that the language expresses requirements purely in terms of the evolution of objects, the requirements themselves remain independent from the implementation of services. As such, the same requirements can be used with various implementations of a conceptual service once these implementations are properly annotated with object evolution information.

5.4 User Protocol

The user protocol of the composite service should give the user enough control over the execution of component services. We distinguish three main types of user control activities that have to be reflected in the user interface and protocol of the composite service. They are:

1. **Execution activation**. Each controllable transition in an object diagram can be triggered through the execution of a particular service. We extend the user interface of the composite service with additional operations that let the user control the activation of each controllable transition (Fig. 2(a)). To activate a certain transition (i. e., to ask the composite service to execute the corresponding service), the user calls special operation $call_i$. If the composite service needs to trigger some controllable transition itself (e. g., in order to meet composition requirements) if first has to receive the user's permission for that and only then to proceed. To implement such interaction we add three more operations: ask_i, $accept_i$ and $reject_i$. First, the



(a) $activation$ (b) $reporting$ (c) $decision$

Fig. 2. Interface for user control elements

composite service sends ask_i and then the user replies with $accept_i$ if the permission is granted and $reject_i$ if not. We impose additional restriction which requires that the user may request the execution of a service only if the composition is not busy with any other task.

2. **Event reporting.** Every event that happens to a domain object has to be reported to the user. Therefore, for each event we add operation $event_i$ to the composition interface (Fig. 2(b)). They let the user stay informed about the dynamics of the domain objects.
3. **Decision-making.** If a service protocol features branching, we allow the user to make decisions at any branching point. Particularly, for each branching point we add to the composition interface the following operations: ask_dec_i, dec_1_i, ..., dec_n_i (Fig. 2(c)). At a branching point, the composition sends ask_dec_i and then receives one of possible decisions $dec\_1_i, \ldots, dec\_n_i$.

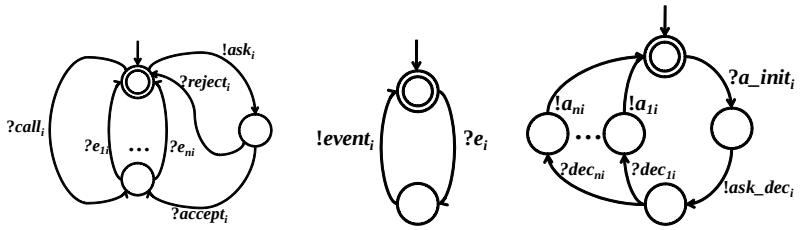
6 Composition as Planning Problem

In order to use planning techniques for composing services, we need to replace a service composition problem with the planning one. More precisely, the formal model has to be transformed into a set of synchronized state transition systems (STS) that form a planning domain, and the composition requirements have to be rephrased in terms of the goal states of these STS's.

In this section, we define transformation rules for each part of our formal model and demonstrate how composition requirements can be expressed as planning goals. We further outline the work of the algorithm and explain how to interpret its results.

Service Transformation. To transform component services we replace a given annotated STS $\langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{E}, \mathcal{R} \rangle$ with a corresponding STS $\langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}', \mathcal{S}^F, \mathcal{F} \rangle$. In order to perform such transformation, for each transition $(s, a, \varepsilon, s') \in \mathcal{R}$ we define a corresponding transition $(s, \top, a, s') \in \mathcal{R}'$, and the states are not labeled (for each $s \in \mathcal{S}\ \mathcal{F}(s) = \emptyset$). In order to state that each service once started has to be executed to one of its final states all the terminating and initial states of the ASTS are marked as accepting states in the STS.

Object diagrams transformation. First, we define a set of atomic propositions $\mathcal{P} \equiv \{s_j{}^{\mathsf{s}}(o_i)\}$, which specify that an object o_i is in state s_j for all objects and their states.



(a) Activation STS (b) Event reporting STS (c) Decision making STS

Fig. 3. STS diagrams of user control elements

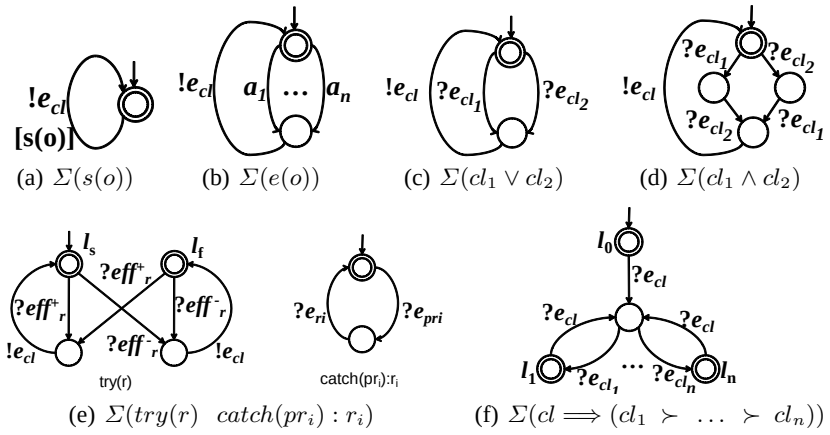
This will allow us to capture object states in requirements. Object diagram $\langle L, L_0, \mathcal{E}, T\rangle$ in transformed into an STS $\langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{S}^F, \mathcal{F}\rangle$, where $\mathcal{S} = L$, $\mathcal{S}^0 = L_0$, every state is labelled with the corresponding proposition (i. e., $\forall s \in \mathcal{S} : \mathcal{F}(s) = \{s^{\mathsf{S}}(o)\}$), and all object configurations are accepting (i. e., $\mathcal{S}^F = \mathcal{S}$). To obtain the transition relation of a new STS, for each non-deterministic transition in the object diagram $(l, c, \{(\varepsilon'_1, l'_1), \ldots, (\varepsilon'_n, l'_n)\}) \in T$ and for any transition (s, a, ε, s') of some ASTS such that $\varepsilon'_i \subseteq \varepsilon$ we define a (non-guarded) transition $(l, \top, a, l'_i) \in \mathcal{R}$.

User Protocol Transformation. For each user control element mentioned in subsection 5.4 we provide an STS implementing its semantics:

- The STS for execution activation (Fig. 3(a)) blocks a corresponding controllable transition in an object diagram (by blocking all events associated $e_{1i} \ldots e_{ni}$) until either the user sends a request ($?call_i$)) or the composite service gets user's authorization to trigger the transition ($accept_i$).
- The STS for event reporting (Fig. 3(b)) sends a report to the user ($!event_i$) every time an associated event ($?e_i$) happens to an object.
- The STS for making decisions (Fig. 3(c)) blocks service execution as soon as a branching point is reached ($?a\_init_i$). It asks the user for a decision ($!ask\_dec_i$) and when the user chooses one of the possible options ($?dec_{1i} \ldots dec_{ni}$) it unblocks a corresponding service action ($!a_{1i} \ldots !a_{ni}$).

Transformation of constraints. The composition constraints speak of both object states and occurrences of object events. For every constraint we define a corresponding STS that reflects the satisfiability of the requirement. Given a clause cl, we define a corresponding STS that contains a single output action e_{cl} representing the completion of the clause. The diagrams corresponding to different clauses and their combinations, and to the constraint diagram itself are represented in Fig. 4.

The STS for $s^{\mathsf{S}}(o)$ (Fig. 4(a)) is blocked until the object is not in the required state: the transition is guarded with the corresponding proposition. We denote the fact that object o is in state s with event e_s. The STS for $e^{\mathsf{e}}(o)$ (Fig. 4(b)) waits for any of the service actions that contain the corresponding event in its effects: for any transition (s, a, ε, s') of some ASTS such that $e^{\mathsf{e}}(o) \in \varepsilon$ a corresponding transition is defined. When it happens, a completion is reported. The STS for $cl_1 \vee cl_2$ (Fig. 4(c)) waits for



(a) $\Sigma(s(o))$ (b) $\Sigma(e(o))$ (c) $\Sigma(cl_1 \vee cl_2)$ (d) $\Sigma(cl_1 \wedge cl_2)$

(e) $\Sigma(try(r) \quad catch(pr_i) : r_i)$ (f) $\Sigma(cl \Longrightarrow (cl_1 \succ \ \ldots \ \succ \ cl_n))$

Fig. 4. STS diagrams of composition constraints

any of the sub-clauses to complete, while the STS of the $cl_1 \wedge cl_2$ (Fig. 4(d)) waits for both of them to be completed.

The transformation of the $try - catch$ construction (Fig. 4(e)) includes a try-STS and one STS for each $catch$ block. For each $catch(pr_i) : r_i$ we create an STS enforcing that every failure pr_i is entailed with appropriate reaction r_i. $try(r)$ is reported to be completed when either the needed reaction r has been provided successfully (we denote this situation with event $\mathit{eff}^{+}(r)$) or some fail happened while trying to provide r (event $\mathit{eff}^{-}(r)$). Given a transition $t = (l_0, \{e_i, l_i\})$ of object o let $finals(t) = \{l_i\}$, $events(t) = \{e_i\}$, $to(t, e) = l'$ if $(e, l') \in \{e_i, l_i\}$ and $reject(t)$ corresponds to a situation when permission for triggering t is not granted (transition $reject_i$ in 3(a)), which can be considered as transition failure. Then $\mathit{eff}^{+}(r)$ and $\mathit{eff}^{-}(r)$ can be defined for different reactions as follows:

- $\mathit{eff}^{+}(s^{\mathsf{s}}(o)) = e_s$,
  $\mathit{eff}^{-}(s^{\mathsf{s}}(o)) = \{e' : (\exists t \in T, e'' \in \mathcal{E} : to(t, e'') = s) \wedge (e \in events(t))\} \bigcup$
  $\{reject(t) : t \in T_c \wedge s \in finals(t)\}$;
- $\mathit{eff}^{+}(e^{\mathsf{e}}(o)) = e^{\mathsf{e}}(o)$,
  $\mathit{eff}^{-}(e^{\mathsf{e}}(o)) = \{e' : (\exists t \in T : e \in finals(t) \wedge e' \in finals(t) \wedge e \neq e')\} \bigcup$
  $\{reject(t) : t \in T_c \wedge e \in events(t)\}$;
- $\mathit{eff}^{+}(cl_1 \vee cl_2) = \mathit{eff}^{+}(cl_1) \cup \mathit{eff}^{+}(cl_2)$, $\mathit{eff}^{-}(cl_1 \vee cl_2) = \mathit{eff}^{-}(cl_1) \times \mathit{eff}^{-}(cl_2)$;
- $\mathit{eff}^{+}(cl_1 \wedge cl_2) = \mathit{eff}^{+}(cl_1) \times \mathit{eff}^{+}(cl_2)$, $\mathit{eff}^{-}(cl_1 \wedge cl_2) = \mathit{eff}^{-}(cl_1) \cup \mathit{eff}^{-}(cl_2)$;
- $\mathit{eff}^{+}(try(r)) = \mathit{eff}^{+}(r) \cup \mathit{eff}^{-}(r)$, $\mathit{eff}^{-}(try(r)) = \emptyset$.

Intuitively, these definitions can be explained in the following way. The attempt to reach a state fails when the composition tries to trigger one of the transitions potentially leading to this state but fails. The attempt to cause some event fails when the composition tries to trigger one of the transitions potentially causing this event but fails. The attempt to provide $try(r)$ is always successful (any outcome of the attempt to trigger a transition

is affordable). It is important to notice that $\mathit{eff}^{-}(r)$ for controllable transitions includes additional events that do not appear explicitly in an object diagram. These events correspond to the situations where the user does not authorize the composite service to execute a service (message $?reject_i$ in Fig. 3(a)). Indeed, when the system, according to requirements, is supposed to *try* to do something, but the user does not grant permission to start necessary service execution, we can conclude that the composition has actually tried to react but failed. In try-STS, state l_s is more preferable than state l_f. The corresponding goal with preferences will be:

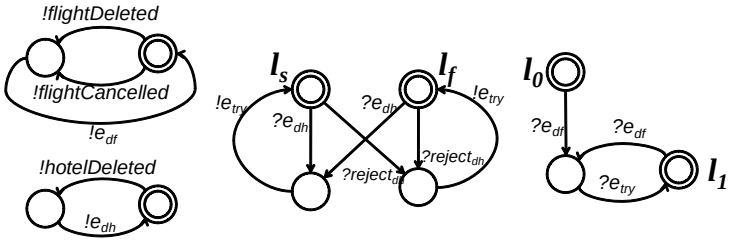
$$\rho_t = (l_s, l_f) \tag{1}$$

The STS that represents the evolution of a composition constraint is represented in Fig. 4(f). The STS is initially in an accepting location (l_0). If the premise of the constraint takes place (e_{cl} is reported), then it moves to a non-accepting state, from which it may be satisfied by completing one of the clauses $e_{cl_1}, \ldots, e_{cl_n}$ (moving to locations $l_1, \ldots, l_n$ respectively). The corresponding goal with preferences will have the following form:

$$\rho_c = (l_0, l_1, \ldots, l_n). \tag{2}$$

That is, we require that whenever the premise take place, the composition tries to move the constraint STS to one of the accepting states, respecting the ordering of preferences.

Example 4. The constraint $d^{\mathsf{e}}(f) \Longrightarrow try(d^{\mathsf{e}}(h))$ is modelled with the following STSs:



Although the hotel cancellation process is deterministic and cannot fail during the execution, negative outcome $\mathit{eff}^{-}(d^{\mathsf{e}}(f))$ for event $d^{\mathsf{e}}(h)$ is not empty because, as we discussed earlier, the user's refusal to grant permission to execute some service (here $reject_{dh}$) is also considered to be a failure.

6.1 Generating a Composite Service

The approach to automated service composition that we use here refers to the one presented in [10]. In particular, having a set of n services ($W_1, \ldots, W_n$), m objects ($O_1, \ldots, O_m$), k composition constraints ($C_1, \ldots, C_k$) and l user control elements ($U_1, \ldots, U_l$) we transform them into STSs using the transformation rules presented above so that we get an STS for each part of the model (respectively $\Sigma_{W_1} \ldots \Sigma_{W_n}$, $\Sigma_{O_1} \ldots \Sigma_{O_m}$, $\Sigma_{C_1} \ldots \Sigma_{C_k}$ and $\Sigma_{U_1} \ldots \Sigma_{U_l}$).

The planning domain Σ is a synchronous product of all the STSs of the component services, objects, constraints and user control elements while composition goal is constructed from the requirements defined according to the formulae (1) and (2):

$$\Sigma = \Sigma_{W_1} \| \ldots \| \Sigma_{W_n} \parallel \Sigma_{O_1} \| \ldots \| \Sigma_{O_m} \parallel \Sigma_{C_1} \| \ldots \| \Sigma_{C_k} \parallel \Sigma_{U_1} \| \ldots \| \Sigma_{U_l}$$

$$\rho = \bigwedge_c \rho_c \wedge \bigwedge_t \rho_t.$$

Finally, we apply the approach of [10] to domain Σ and planning goal ρ and generate a controller Σ_c, which is such that $\Sigma_c \triangleright \Sigma \models \rho$. The state transition system Σ_c is further translated into executable BPEL process, which implements a composite service required. The back translation from STS into BPEL is quite simple: input actions in Σ_c model an incoming message from a component service or from the user, while output actions in Σ_c model an outcoming message to a component service or to the user.

Correctness of the approach. The proof of the correctness of the approach consists in showing that all the executions of the composed service (controller Σ_c) satisfy the control flow requirements defined. Here we outline the key points of the proof. It is easy to see that each execution π of the composed service is also a run of the domain, i. e., if $\pi \in \Pi(\Sigma_c)$ then $\pi \in (\Sigma)$. Under the requirement that all the executions of constraint STS terminate in accepting states, we get that the executions of the domain satisfy the composition requirements. As a consequence, the following theorem holds.

Theorem 1 (Correctness of the approach). *Let:*

- $\Sigma_{W_1}, \ldots, \Sigma_{W_n}$ *be the STS encoding of component services* $W_1, \ldots W_n$,
- $\Sigma_{C_1}, \ldots, \Sigma_{C_k}$ *be the STS encoding of the constraints* $C_1, \ldots C_k$,
- $\Sigma_{O_1}, \ldots, \Sigma_{O_m}$ *be the STS encoding of the domain objects* $O_1, \ldots, O_m$, *and*
- $\Sigma_{U_1}, \ldots, \Sigma_{U_l}$ *be the STS encoding of the user control elements* $U_1, \ldots, U_l$.

Let Σ_c *be the controller for a particular composition problem*

$$\Sigma = \Sigma_{W_1} \| \ldots \| \Sigma_{W_n} \parallel \Sigma_{O_1} \| \ldots \| \Sigma_{O_m} \parallel \Sigma_{C_1} \| \ldots \| \Sigma_{C_k} \parallel \Sigma_{U_1} \| \ldots \| \Sigma_{U_l}.$$

Then the executions $\Pi(\Sigma_c)$ *satisfy the requirements* $C_1, \ldots, C_k$.

6.2 Experiments

We evaluate the potential of our approach by implementing the preliminary version of the composition framework and testing it with a reference case study. Having service specifications, object diagram descriptions, composition requirements and user control elements the composition framework generates a planning domain specification with the goals defined over this domain. After the planning algorithm is applied to the planning problem, the plan, which encodes the expected behaviour of the composite process, is translated back into BPEL.

The composed process implements the composition requirements presented. The process first waits for the request to create a flight ticket or a hotel reservation and

then transactional books both items processing fails when happened. After both object are created, if the flight is delayed or cancelled, the appropriate reaction follows. The interaction with the user is embedded into the composite process. In particular, at stable point of the execution, where all the requirements are satisfied, the user is allowed to request certain operations on objects (i.e., service executions). Whenever some event happens, it is reported to the user. Finally, at branching points in service protocols, the composition asks for the user's decision and takes it into account later on.

As we shown in Fig. 1, the implementations of component services have fairly complex structure with 14 BPEL activities (such as receive, response etc.) in the most complex process. However, the composed executable process composed is much more complicated, with around 100 BPEL activities, including those providing interaction with the user. The process features very complex structure with multiple branching points and several loops. No doubt, its manual coding would take a significant amount of time.

The run of the experiment took less than 15 seconds on a Windows-machine with dual core 2 GHz CPU and 4Gb of memory. Taking into account further possible optimization, we consider these results to be very promising.

7 Related Work and Conclusions

In this paper, we have presented a novel approach to modelling and composing user-centric services. Being based on the notion of domain objects, the approach features an expressive language for control-flow requirements allowing for continuous orchestration of services using reactive rules and coordination constraints. The language is detached from service implementations and expresses the requirements in terms of domain object evolution. The approach also provides the automatic derivation of the user protocol, which enables the user to control the composition execution and to be aware of the current execution progress. While some parts of the composition model have to be specified manually, the model resolution, which is the most time and effort demanding phase of the composition process, is completely automated using planning techniques. The paper also provides some preliminary experimental results, which prove the potential of our approach.

The new model proposed in this paper is a further development of the model of [4]. Our composition engine relies on the service composition via automated planning presented in [10]. In this paper, we focused on control-flow composition requirements and did not consider data-flow composition requirements. We plan to integrate data-flow requirements as future work, exploiting the approach described in [8], which is also based on [10] and is hence compatible with our approach.

There is a wide literature on service composition. Most of the proposed approaches, however, model services as atomic entities (e.g., [1], [12] and [13]), while we consider services to be non-deterministic, stateful and asynchronous. In particular, while there are some other user-centric services composition approaches (e.g., [6], [9], [5]), they give a very different meaning to "user-centricity" than the one we used in this paper, and they address problems that are quite different from ours. The new language for control-flow requirements and the ability to derive complex user protocol also make our approach different from [10], [4], [3] and [7]. The representation of services by more

abstract "object" entities is close to WS-Conversation language [2]. To some extent, our composition model is closer to the concept of mash-ups [15]. However, mash-ups are usually exploited for integration of data and of information services, and the extension to the full-fledged services considered here is not trivial. The mash-up approach described in [14] is more related to ours. There, mash-up techniques are exploited to guarantee that the coordination is maintained among the status of a set of components to be integrated, and that changes in one of the components is reflected in the others. A key difference is that in [14] the integration and coordination is done at the presentation layer (i.e., user interface), while our approach addresses the problem at the application layer. A better comparison, and possible integration, of the two approaches is part of our plan to extend out composition approach to cover the presentation layer.

Acknowledgement. The research leading to these results has received funding from the European Community Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

1. R. Aggarwal, K. Verma, J. A. Miller, and W. Milnor. Constraint Driven Web Service Composition in METEOR-S. In *Proc. of SCC'04*, pages 23–30, 2004.
2. A. Banerji, C. Bartolini, and D. Beringer. Web Services Conversation Language (WSCL) 1.0. http://www.w3.org/TR/wscl10/, 2002.
3. D. Berardi, D. Calvanese, G. D. Giacomo, and M. Mecella. Composition of Services with Nondeterministic Observable Behaviour. In *Proc. ICSOC'05*, 2005.
4. P. Bertoli, R. Kazhamiakin, M. Paolucci, M. Pistore, H. Raik, and M. Wagner. Control flow requirements for automated service composition. In *ICWS*, pages 17–24, 2009.
5. M. Chang, J. He, W. Tsai, B. Xiao, and Y. Chen. Ucsoa: User-centric service-oriented architecture. *E-Business Engineering, IEEE International Conference on*, 0:248–255, 2006.
6. A. Corradi, E. Lodolo, S. Monti, and S. Pasini. A user-centric composition model for the internet of services. In *ISCC*, pages 110–117, 2008.
7. R. Hull. Web Services Composition: A Story of Models, Automata, and Logics. In *Proc. of ICWS'05*, 2005.
8. A. Marconi, M. Pistore, and P. Traverso. Specifying Data-Flow Requirements for the Automated Composition of Web Services. In *Proc. SEFM'06*, 2006.
9. T. Nestler, L. Dannecker, and A. Pursche. User-centric composition of service front-ends at the presentation layer. In *1st International Workshop on User-generated Services (UGS2009)*, 2009.
10. M. Pistore, P. Traverso, and P. Bertoli. Automated Composition of Web Services by Planning in Asynchronous Domains. In *Proc. ICAPS'05*, 2005.
11. D. Shaparau, M. Pistore, and P. Traverso. Contingent planning with goal preferences. In *Proc. AAAI'06*, 2006.
12. S. Thakkar, J.-L. Ambite, and C. Knoblock. A data integration approach to automatically composing and optimizing web services. In *Proceedings of 2004 ICAPS Workshop on Planning and Scheduling for Web and Grid Services*, Whistler, BC, Canada, 2004.
13. D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition using SHOP2. In *Proc. ISWC'03*, 2003.
14. J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera. A framework for rapid integration of presentation components. In *Proc. WWW'07*, 2007.
15. N. Zang, M. B. Rosson, and V. Nasser. Mashups: who? what? why? In *Proc. CHI'08*, 2008.