Process Reliability in Service Oriented Architectures

Dieter Schuller, Apostolos Papageorgiou, Stefan Schulte, Julian Eckert, Nicolas Repp, Ralf Steinmetz
Technische Universität Darmstadt
Multimedia Communications Lab
Merckstr. 25, 64285 Darmstadt, Germany
{schuller, papageorgiou, schulte, eckert, repp, steinmetz}@KOM.tu-darmstadt.de

***Abstract*—The creation of business processes by composing Web services has attracted considerable efforts with respect to integrating elements of Service Oriented Architectures within enterprise applications. If one of those atomic services fails, the whole business process does not necessarily have to fail, too, as it is possible to exchange the faulty service during runtime with another one which has the same functionality. In this paper we present a service monitoring and exchanging approach that aims to ensure a successful execution of crucial processes in case of faulty services. It consists of three parts which supervise the process execution and try to recover it if an error occurs. In case process reliability is not fore grounded, our approach can also serve as basis for process optimization.**

***Index Terms*—reliability, monitoring, replanning.**

I. Introduction

Nowadays, a certain share of (business) processes is no longer only executed within a single enterprise (cf. [1]–[3]). Processes, which are interlinked with each other and reach far beyond the borders of a single company, become more important (cf. [4]). They are assembled to bigger processes with a more complex functionality. If one of the subordinate processes fails, the risk arises that the task of the overall process is not accomplished successfully. In case this overall process is a critical process, its successful termination has to be guaranteed. Therefore, process reliability has to be considered.

With respect to monolithic software applications, a single error might cause the execution of the whole application to stop. This can be expressed in various forms, ranging from a single error message that something is going wrong to a complete system crash. In the former case the impact of such a software error is not that crucial, but in the latter case serious harm could occur with respect to, e.g., life-supporting systems in a hospital.

When we think of a process that requires to be executed successfully, the emergence of faults or errors leads to the need for correcting the application. This might in turn force the IT-department to adapt it. First, the bug has to be found. Secondly, some IT-specialists have to change the code of the software program. And that is all taking place while the application should deliver the expected results to finish the process successfully.

This can be avoided by having alternative software solutions that substitute the faulty ones immediately. In an Service Oriented Architecture (SOA), mentioned monolithic software applications are replaced by more or less coarse-grained services that are composed to composite services with a more complex functionality (cf. [5]). Processes can be built using services. Once such a process is engineered inside an SOA, there is no need to adjust any software programs during runtime if something goes wrong. As services are loosely coupled, it is possible to replace faulty services with others having the same functionality – if available, for instance, on a market place.

To realize this, we have to notice emerging faults not just by an unexpected abortion of the process but in time, i.e. when a service terminates unsuccessfully or does not terminate at all. That is why service monitoring is indispensable. When the faulty service is identified, an adequate alternative is needed in order to replace the former one. In the paper at hand, we present an approach which therefore covers monitoring, semantic matchmaking, and replanning in order to ensure process reliability.

According to this, the rest of the paper is organized as follows: In Section II we describe our governance scenario in detail and give an overview of our application domain. Thereafter, we explain the components needed to realize such a scenario. In Section III the Monitoring environment is described. Section IV deals with semantic matchmaking in order to find alternative services. In Section V, the arising optimization problem is formulated and the recovering of the process execution in case of failures is discussed. Related work is presented in Section VI. Finally, conclusions are drawn and future work is discussed in Section VII.

II. Governance Scenario

We consider process reliability in the context of the project SoKNOS. In this project we come from a catastrophe scenario. Heavy rains cause a crevasse that threatens a hospital and a chemical plant. Therefore the hospital has to be evacuated and the chemical plant has to be secured in order to avoid an ecological disaster. The responsible staff has to be provided with the most relevant, accurate and precise information about the situation. Based on this information, they work out and communicate countermeasures. Afterwards, it is necessary to check the enforcement of the agreed countermeasures and its effects on the situation.

These tasks are encapsulated in multiple processes. Not only one single office is participating but a lot of departments are involved. In the context of the aforementioned project, at least firefighters, the THW (Technisches Hilfswerk – a German aid

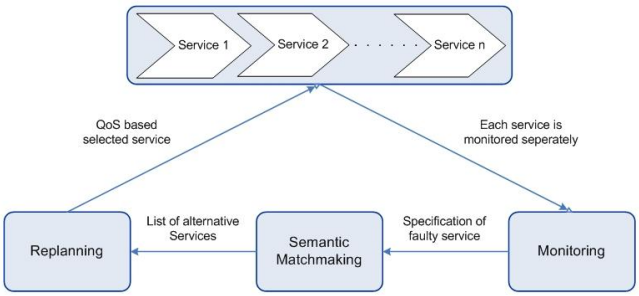

Fig. 1. Monitoring-Replanning-Cycle

organization), police, departments providing geological data, and departments coordinating the necessary workflows are participating. These coordinating departments have to provide all the other parties with information relevant to their tasks, respectively, and receive feedback to control their activities. So, on the one hand, a lot of information is exchanged. On the other hand, the operating units themselves might make use of IT systems to support their workflows. In both cases, software applications of many kinds are employed.

In this paper we focus on critical processes, keeping the introduced catastrophe scenario in mind. The failure of one process might cause other processes to fail, too. This decreases the efficiency of the decided countermeasures. In worst-case, necessary operations cannot be accomplished, which leads to mishandling of the aforementioned ecological disasters or emergency situations. Efficient replacement of faulty services reduces the risk of unsuccessful termination of such critical processes.

Therefore, we have implemented a SOA governance approach which considers service replacement. The main idea for this is summarized in Fig. 1.

The Monitoring environment diagnoses if a service fails or malfunctions and informs the Semantic Matchmaking component to find alternative services. Receiving the specification of the faulty service, the Semantic Matchmaking component identifies and delivers the requested possible alternative services to the Replanning component, which creates an optimal execution plan, utilizing the received list of alternative services, in order to finish the non-accomplished tasks.

III. Monitoring

All actions of semantic substitution or replanning are actually triggered by the monitoring component, whenever the latter detects such a need during the system operation. Different approaches for the monitoring of services inside an SOA have appeared. [6] is an agent-based solution supported by a service proxy. The cooperation of a proxy module with JADE agents, which perform the actual monitoring, offers to the system a seamless monitoring layer. In another proposal by [7], a "diagnostic service" uses data not only from the monitoring service of the proposed architecture but also from host and network monitoring tools, in order to diagnose QoS

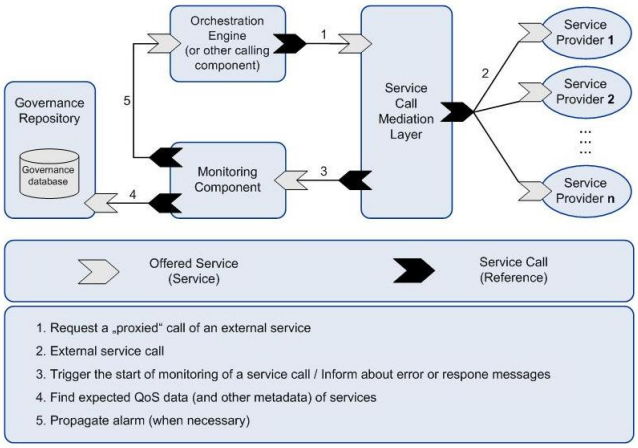

Fig. 2. Monitoring Environment

anomalies. The monitoring approach presented in this section has been steered mainly by the following two needs:

- Fit its role as one component of our three-dimensional governance system.
- Fit the architecture of the system on which it is being applied and take advantage of its features.

How the above needs dictated the current solution will be better understood in the following, where the architecture of the monitoring component and its interactions with the other components are described.

In accordance with the aforementioned needs, two features shape the architecture of the monitoring component. First, the triggering of service substitution and replanning must be real-time and event-driven. This means, when the monitoring component detects a failure or unusual delay, we start semantic matchmaking and replanning immediately. In order to be more efficient, we consider in our future work to start semantic matchmaking and replanning at the same time the service execution begins. Secondly, the monitoring component must exploit the functionality of the external service mediation layer, a component of our system, which actually works as a proxy for external service calls.

So, when the mediation layer starts the execution of an external service, the monitoring component is informed, i.e. it starts the actual monitoring (cf. Fig. 2). Any response or fault caught by the mediation layer is sent automatically to the monitoring component. So, either we receive an error notification or we notice a Service Level Agreement (SLA) violation by comparing the actual performance of the current service with the performance committed in its SLAs. In both cases, the monitoring component generates an alarm which is propagated to the Semantic Matchmaking component.

It is critical for the flexibility and the extensibility of the system that all components follow the principles of service-orientation. To this end, all of the described component-actions and inter-component communication are performed with Web services, as well. For example, the Governance Repository, which contains service IDs, SLAs, average response times etc.,

offers a set of services allowing the monitoring component to access this data. Similarly, the monitoring module offers a set of services to trigger the monitoring start and the notifications. In our scenario, the latter are called by the mediation layer. Even the alarms generated by the monitoring component are propagated using Web services, offered this time by the components that are "interested" in the alarms. The usage of all these services is not limited in the described monitoring scenario. According to the principle of service reuse, they are involved in other functionalities, as well. For example, the services offered by the Governance Repository are also employed by GUIs for the assistance of manual governance.

IV. Semantic Matchmaking

After the failure of a service has been detected, it is necessary to retrieve services which are able to provide the same functionality as the failed service. There are two possibilities on how to substitute a service – either there is a market for this service [8], i.e. a similar service is publicly available, or a substitute is provided inside our own domain. In both cases, it is necessary to clearly depict the functionalities of all services which come into consideration as substitutes in order to facilitate the automatic replanning of services and workflows. Here, service providers and service requestors rely on an ontology which describes the service domain. Using such an ontology, it is possible to annotate the service descriptions with semantic information, i.e. information which is especially needed if it is necessary to develop intelligent Internet applications which are easy to retrieve, access and use [9].

While in many domains well-defined ontologies are still missing [10], the scenario at hand provides an ontology which can be used in order to identify the functionalities of services. Thus, the semantic data in the service description specifies the functionalities of each service and can be used to identify similar services. The finding of substitute Web services is straightforward as we can act on the assumption that we possess complete semantic information about the functionalities of each service candidate. Furthermore, we describe the QoS aspects, which will be needed in the next step (cf. Section V) also semantically, using an extended version of WS-Re2Policy, a language which allows the description of QoS requirements as well as countermeasures to be triggered if the requirements cannot be met [11].

If a service fails, we use its service description as an input for the service retrieval engine. Using a reasoner, the engine provides a number of possible service candidates, which offer the same functionality but different QoS levels. In the next step, it is necessary to compare and assess these levels and pick the best-fitting service (regarding our own QoS requirements) out of the set of service candidates.

V. Replanning

In the context of processes based on Web services, replanning deals with changing the execution plan of a process due to SLA deviations – these deviations can be positive or negative – in order to obtain an optimal process execution (cf. [12]–[14]). Optimal thereby means to select the best services with respect to QoS properties which comply with the given SLA restrictions. A positive (negative) deviation is at hand when a service performed better (worse) than committed in its SLAs. In both cases, it is possible to make a target-performance comparison in order to adapt the process conditions for the remaining services and create a new optimal process execution for the remaining tasks with the adjusted process conditions. I.e., if we consider a process which is composed of five services and needs to be successfully executed in 10 seconds, the sum of the assured execution times of the five services must not exceed 10 seconds. If the first service performs one second faster than assured in its SLA, the allowed execution time for the rest of the services comes up with this one second in addition. Therefore, we could select other services with probably higher execution times at lower prices. To find optimal or near optimal solutions for the service selection problem, many approaches have been proposed (cf. [15]–[18]). Our approach is based on the considerations by [19], applied to SoKNOS project, and further extended.

To create the system model and formulate the optimization problem we assume a sequential execution of n tasks, which forms only a subset of possible workflows. Queuing theory allows complex networks, which lead to complex workflows containing cycles, branches, joins, etc. to be analyzed, too [20]. Task $i \in I = \{1, ..., n\}$ is executed before task $i+1$. For each task i, m_i alternative Web services $j_i \in J_i = \{1, ..., m_i\}$ exist, whereby task i is accomplished by exactly one service. These Web services might differ in o_i QoS attributes $k_i \in K_i = \{1, ..., o_i\}$ that are weighted with weights w_{ik}, respectively. Thereby, we propose $\sum_{k_i=1}^{o_i} w_{ik} = 1, \forall i \in I$. The variables a_{ijk} represent the values of the QoS attributes. If a higher (lower) value indicates a better QoS property, we define this QoS attribute to be positive (negative). Depending on this introduced positive and negative QoS property, we normalize the QoS values in equations (1) and (2) in a way that higher values of these attributes indicate higher quality. The SLA conditions for the different QoS types are normalized analogously in (3).

$$a^s_{ijk} := \begin{cases} 1 - \frac{a_{ijk} - min\{a_{ijk}\}}{a_{ijk}} & \text{, if negative QoS} \\ \frac{a_{ijk}}{max\{a_{ijk}\}} & \text{, else} \end{cases} \quad (1)$$

$$a^{adj}_{ijk} := \begin{cases} -a_{ijk} & \text{, if negative QoS} \\ a_{ijk} & \text{, else} \end{cases} \quad (2)$$

$$r^{adj}_{ik} := \begin{cases} -r_{ik} & \text{, if negative QoS} \\ r_{ik} & \text{, else} \end{cases} \quad (3)$$

The decision variables $x_{ij} \in \{0, 1\}$ state, whether Web service j of the alternative services for task i is selected. So, the initial optimization problem is depicted in Model 1. By setting values for the weights w_{ik}, the target function can totally be adjusted to the needs of the user. So, a high weight

for one or more QoS attributes indicates that we would rather prefer those services that have high values for these QoS attributes.

Model 1 Linear Optimization Problem

Target function

$$\text{maximize } F(x) = \sum_{i=1}^{n} \sum_{j=1}^{J_i} \sum_{k=1}^{K_i} w_{ik} a^s_{ijk} x_{ij} \tag{4}$$

s.t.

$$\sum_{j=1}^{J_i} a^{adj}_{ijk} x_{ij} \geq r^{adj}_{ik} \qquad \forall i \in I, \forall k_i \in K_i \tag{5}$$

$$\sum_{j=1}^{J_i} x_{ij} = 1 \qquad \forall i \in I \tag{6}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in I, \forall j \in J_i \tag{7}$$

As our goal is to optimize process execution we have to calculate the values of the QoS attributes for the overall process. Therefore, we have to differentiate at least the three calculation types summation, multiplication and min/max operator. For the sake of clarity, we will refer to "min" in our equations and define the min/max operator in (8), (9).

$$min\{x_i\} := Minimum\{x_i\} \tag{8}$$

$$max\{x_i\} := Maximum\{x_i\} \tag{9}$$

With respect to its type we label the QoS attributes for the services accomplishing task i with $k^+_i \in K^+_i = \{1, ..., o^+_i\}$, $k^*_i \in K^*_i = \{1, ..., o^*_i\}$, and $k^{min}_i \in K^{min}_i = \{1, ..., o^{min}_i\}$ and postulate $o_i = o^+_i + o^*_i + o^{min}_i$. We further adapt (2) to (10) to (12).

$$a^+_{ijk} := \begin{cases} -a_{ijk} & \text{, if negative QoS} \\ a_{ijk} & \text{, else} \end{cases} \tag{10}$$

$$a^*_{ijk} := \begin{cases} \frac{1}{a_{ijk}} & \text{, if negative QoS} \\ a_{ijk} & \text{, else} \end{cases} \tag{11}$$

$$a^{min}_{ijk} := \begin{cases} -a_{ijk} & \text{, if negative QoS} \\ a_{ijk} & \text{, else} \end{cases} \tag{12}$$

To sum up the QoS values over all tasks, we have to assure that the services accomplishing tasks i have the same amount of QoS attributes. This is not necessarily the case in reality as the tasks differ from each other. Therefore, we fill up the missing QoS attributes with zeros, ones or infinite, depending on its summation types, respectively. In order to avoid confusing the reader, we point to the index i that is partly missing in the following definitions and formulas. We define o as $o = max\{o_i\}$. Consequently, we define $k \in K = \{1, ..., o\}$ and label the QoS attributes with respect to its types with $k^+ \in K^+ = \{1, ..., o^+\}$, $k^* \in K^* = \{1, ..., o^*\}$, and $k^{min} \in K^{min} = \{1, ..., o^{min}\}$ with $o = o^+ + o^* + o^{min}$. Further, we need conditions for the QoS attributes of the whole process. Thus, we adapt the values with respect to the different QoS types in (13) to (15) and specify process restrictions in (16) to (18) here fore.

$$p^+_k := \begin{cases} -p_k & \text{, if negative QoS} \\ p_k & \text{, else} \end{cases} \tag{13}$$

$$p^*_k := \begin{cases} \frac{1}{p_k} & \text{, if negative QoS} \\ p_k & \text{, else} \end{cases} \tag{14}$$

$$p^{min}_k := \begin{cases} -p_k & \text{, if negative QoS} \\ p_k & \text{, else} \end{cases} \tag{15}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{J_i} a^+_{ijk} x_{ij} \geq p^+_k \qquad \forall k^+ \in K^+ \tag{16}$$

$$\prod_{i=1}^{n} \sum_{j=1}^{J_i} a^*_{ijk} x_{ij} \geq p^*_k \qquad \forall k^* \in K^* \tag{17}$$

$$min\{\sum_{j=1}^{J_i} a^{min}_{ijk} x_{ij}\} \geq p^{min}_k \qquad \forall k^{min} \in K^{min} \tag{18}$$

In the context of our scenario, we stress on fast and successful process execution by assigning high weights for certain QoS attributes as, e.g., for availability, reliability, and execution time. So, we aim to solve the optimization problem depicted in Model 2.

Model 2 Non-linear Optimization Problem

Target function

$$\text{maximize } F(x) = \sum_{i=1}^{n} \sum_{j=1}^{J_i} \sum_{k=1}^{K_i} w_{ik} a^s_{ijk} x_{ij} \tag{19}$$

s.t.

$$\sum_{i=1}^{n} \sum_{j=1}^{J_i} a^+_{ijk} x_{ij} \geq p^+_k \qquad \forall k^+ \in K^+ \tag{20}$$

$$\prod_{i=1}^{n} \sum_{j=1}^{J_i} a^*_{ijk} x_{ij} \geq p^*_k \qquad \forall k^* \in K^* \tag{21}$$

$$min\{\sum_{j=1}^{J_i} a^{min}_{ijk} x_{ij}\} \geq p^{min}_k \qquad \forall k^{min} \in K^{min} \tag{22}$$

$$\sum_{j=1}^{J_i} x_{ij} = 1 \qquad \forall i \in I \tag{23}$$

$$x_{ij} \in \{0,1\} \quad \forall i \in I, \forall j \in J_i \tag{24}$$

As we have non-linear terms in our equations, too, the optimization problem cannot be solved using linear programming

techniques. So, we use the approximation in (25) that is very accurate with parameter values z_{ij} close to one. Using this approximation, we exchange (21) by (26), and relax (22) to (27).

$$\prod_{i=1}^{n}\sum_{j=1}^{J_i} z_{ij}x_{ij} \approx 1 - \sum_{i=1}^{n}(1 - \sum_{j=1}^{J_i} z_{ij}x_{ij}) \tag{25}$$

$$1 - \sum_{i=1}^{n}(1 - \sum_{j=1}^{J_i} a^*_{ijk}x_{ij}) \geq p^*_k \qquad \forall k^* \in K^* \tag{26}$$

$$\sum_{j=1}^{J_i} a^{min}_{ijk}x_{ij} \geq p^{min}_k \qquad \forall i \in I, \forall k^{min} \in K^{min} \tag{27}$$

In order to calculate an execution plan, Zeng et al. [21] propose integer programming. Depending on the amount of tasks and the amount of candidate services for each task, the computation of the optimal solution probably requires strong computational effort with a growing number of tasks and alternative services, as the optimization problem is NP-hard. Therefore, we relax the integrality conditions and use mixed integer linear programming techniques to calculate the optimal solution. Afterwards, heuristics can be applied to obtain a valid one, which contains integer values for the decision variables. As a possibility, H1_RELAX_IP designed by Berbner et al. [19] could be applied. This heuristic does not perform significantly worse compared with the optimal solution (cf. [22]).

In the context of our scenario, we neither assume to have a huge amount of tasks nor of candidate services. Thus, the exact calculation of the optimal solution using linear programming techniques only takes little time. With respect to our project scenario, we give attributes like availability, reliability and execution time a much higher weight then, e.g., price. In fact, we here fore set the weight to zero. After identifying the optimal solution, the process execution begins. If a service performed better than committed in its SLAs (positive SLA deviation), we could execute a beneficial replanning of the successive services. I.e., as QoS constraints of the process are adjusted by the positive SLA deviation of the successfully executed service, it is possible to exploit these new constraints by solving the optimization problem for the remainder of the tasks. We surely could do this in order to keep the execution plan optimal, but in the context of our project, we primarily focus on error recovery. So, we do not calculate a new optimal solution for positive SLA deviation. We only operate, when a service fails. On service failure at task i, the semantic matchmaking delivers a list of candidate services to the replanning component, which replaces its former list of services for task i – that has been used to solve the optimization problem initially – with the current delivered one. Afterwards, the process condition concerning process execution time is adjusted by the time it took the monitoring and the semantic matchmaking component to detect the SLA deviation and provide the replanning component with the mentioned list of candidate services. With this adjusted condition for the allowed remaining execution time and the list of current possible alternative services, the optimization problem is solved again – not starting at task 1 but at task i – in order to determine an optimal solution for the rest of the originally process.

To make the above workflow more efficient in case of an emergency, we consider, as mentioned, monitoring, semantic matchmaking, and replanning already to begin by starting the execution of a service. So, we do the described actions in parallel to the service execution in order to realize error recovery more quickly.

VI. Related Work

As our approach includes the cooperation of three aspects that could be handled separately, we are interested in related work concerning all of the three fields. The approaches closest to ours, or the ones on which our work is based, have already been described in the corresponding sections.

In [23] a reflective middleware approach called SOAR (SOA with reflection) is proposed to establish a dependable SOA. The authors state that dynamic binding of services is insufficient to achieve a high dependability of an SOA. Further, the dependability assurance for an SOA has to be dynamic and automated. A reflective system is defined as a two-level computational system. Base entities performing usual system functionality form the base level whereas meta-entities performing reflection on the system form the meta-level. As a meta-model, Huang et al. [23] introduce the two-level meta-model SOAR, whereas amongst others first level meta-objects are monitored and adapted by one second-level meta-object. In our work, we realized an approach for monitoring and adapting a SOA system, which is essential to a dependable SOA according to [23].

Concerning monitoring, the idea for "proxied" service calls in [6] and the existence of a diagnosing environment introduced in [7] can be compared with our proposed solution, as described in Section III. In [24], the authors do not present a monitoring architecture, but rather focus on the model-driven development of Web service compositions, i.e., they present tools and steps referring to the use of UML-like meta-models for the design and creation of Web service compositions. However, in focusing on the development of monitored Web service compositions, the presented model differs from others.

Anselmi et al. [25] present in their work a QoS broker-based framework. A broker handles Web service composition requests by selecting appropriate Web services based on required QoS. Thereby, not only one workflow (as we focused on in this paper) but multiple workflows consisting of various tasks, which can be accomplished by services, are optimized. The authors implement a heuristic in which the optimal solution for one optimization problem is used as basis for the solution of the other Web service composition requests to cope with the thereby arising computational effort. But, with respect to building an optimal execution plan for one single process, Anselmi et al. neither consider positive nor negative SLA

deviations in order to keep the process execution optimal and valid.

Besides the described performance analysis, also average-case and worst-case consideration have to be considered as well (cf. [26], [27]).

VII. Conclusion and Outlook

In the paper at hand, we presented parts of a governance approach which focus on process reliability in SOAs. Therefore, we developed the described components Monitoring, Semantic Matchmaking, and Replanning, which is not implemented yet. If the considered process is not critical, our approach equally addresses process optimization.

Our future work aims at proving the feasibility of the presented approach, developing new heuristics here fore, and thereby considering more complex task flows. We further aim to optimize not only single, independent processes but multiple processes keeping the results of Anselmi et al. in mind. In the context of our scenario, not only processes for one department but for all involved departments will be considered.

Acknowledgements

This work is supported in part by the BMBF-sponsored project SoKNOS (http://www.soknos.de) and the E-Finance Lab e. V., Frankfurt am Main, Germany. (http://www.efinancelab.com).

References

[1] C. Bussler, "The role of b2b protocols in inter-enterprise process execution," in *In TES 01: Proceedings of the Second International Workshop on Technologies for E-Services*. Springer-Verlag, 2001, pp. 16–29.

[2] Q. Chen, Q. Chen, M. Hsu, and M. Hsu, "Inter-enterprise collaborative business process management," in *In Proc. of 17th Int. Conference on Data Engineering (ICDE*. IEEE Computer Society, 2001, pp. 253–260.

[3] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana, "The next step in web services," *Commun. ACM*, vol. 46, no. 10, pp. 29–34, 2003.

[4] F. Leymann and D. Roller, *Production workflow: concepts and techniques*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000.

[5] D. Krafzig, K. Banke, and D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[6] N. Repp, "Monitoring of services in distributed workflows," in *3rd International Conference on Software and Data Technologies, INSTICC PRESS, Portugal*, July 2008, pp. 15–25.

[7] G. Wang, C. Wang, A. Chen, H. Wang, C. Fung, S. Uczekaj, Y.-L. Chen, W. Guthmiller, and J. Lee, "Service level management using qos monitoring, diagnostics, and adaptation for networked enterprise systems," in *EDOC Enterprise Computing Conference*, September 2005, pp. 239–248.

[8] M. P. Papazoglou, "Service-oriented computing: Concepts, characteristics and directions," in *Proceedings of the 4th International Conference on Web Information Systems Engineering, IEEE Computer Society, Washington, DC*, 2003, pp. 3–12.

[9] E. Hyvoenen, "Semantic web kick-off in finland: Vision, technologies, research, and applications." in *HIIT Publications 2002-01, Helsinki, Finland*, 2002.

[10] M. Hepp, "Possible ontologies how reality constrains the development of relevant ontologies," *IEEE Internet Computing*, vol. 11, pp. 90–96, 2007.

[11] N. Repp, A. Miede, M. Niemann, and R. Steinmetz, "WSRe2Policy: A policy language for distributed SLA monitoring and enforcement," in *Proceedings of the Third International Conference on Systems and Networks Communications, I. C. Society*, October 2008.

[12] M. C. Jaeger and H. Ladner, "Improving the qos of ws compositions based on redundant services," in *Proceedings of the International Conference on Next Generation Web Services Practices, IEEE*, 2005.

[13] J. F. Zhang, X. T. Nguyen, and R. Kowalczyk, "Graph-based multiagent replanning algorithm," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, 2007.

[14] R. Gronmo and M. C. Jaeger, "Model-driven methodology for building qos-optimised web servce compositions," *International Federation for Information Processing*, vol. 3543/2005, pp. 68–82, 2005.

[15] G. Canfora, M. Penta, R. Esposito, and M. L. Villani, "Qos-aware replanning of composite web services," in *ICWS 2005 Proc.*, 2005.

[16] Cardoso, "Quality of service and semantic composition of workflows," Ph.D. dissertation, University of Georgia, 2002.

[17] D. B. Claro, P. Albers, and J. K. Hao, "Selecting web services for optimal composition," in *ICWS 2005 Workshop Proc., Orlando*, 2005.

[18] T. Yu and K. J. Lin, "Service selection algorithms for composing complex services with multiple qos constraints," in *In ICSOC 2005 Proc., Amsterdam*, 2005.

[19] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Heuristics for qos-aware web service composition," in *IEEE International Conference on Web Services, Chicago, USA*, 2006.

[20] B. R. Harverkort, *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons Inc., New York, 1998.

[21] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnamam, and H. Chang, "Qos-aware middleware for web services composition," in *IEEE Trans. on Soft., England*, May 2004.

[22] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz, "Dynamic replanning of web service workflows," in *Digital EcoSystems and Technologies Conference, Inaugural IEEE-IES*, 2007.

[23] G. Huang, X. Liu, and H. Mei, "SOAR: towards dependable Service-Oriented Architecture via reflective middleware," *Int. J. Simulation and Process Modelling*, vol. 3, no. 1/2, pp. 55–65, 2007.

[24] C. Momm, T. Detsch, and S. Abeck, "Model-driven instrumentation for monitoring the quality of web service compositions," in *IEEE/IFIP Network Operations and Management Symposium, Salvador da Bahia, Brazil*, 2008.

[25] J. Anselmi, D. Ardagna, and P. Cremonesi, "A qos-based selection approach of autonomic grid services," in *Proceedings of the 2007 workshop on Service-oriented computing performance: aspects, issues, and approaches*, 2007, pp. 1–8.

[26] J. Eckert, S. Schulte, N. Repp, R. Berbner, and R. Steinmetz, "Queuing-based capacity planning approach for web service workflows using optimization algorithms," in *IEEE International Conference on Digital Ecosystems and Technologies 2008 (IEEE DEST 2008)*, Feb 2008.

[27] J. Eckert, S. Schulte, M. Niemann, N. Repp, and R. Steinmetz, "Worst-case workflow performance optimization," in *3rd International Conference on Internet and Web Applications and Services (ICIW'08), Athens, Greece*, Jun 2008.