Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2007 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2007

An Approach for Capacity Planning of Web Service Workflows

Julian Eckert
Technische Universität Darmstadt

Nicolas Repp
Technische Universitaet Darmstadt

Stefan Schulte
TU Darmstadt

Rainer Berbner
Technische Universität Darmstadt

Ralf Steinmetz
Technische Universität Darmstadt

AN APPROACH FOR CAPACITY PLANNING OF WEB SERVICE WORKFLOWS

Julian Eckert, Nicolas Repp, Stefan Schulte, Rainer Berbner, Ralf Steinmetz
Department of Computer Science
Technische Universität Darmstadt, Germany
{eckert, repp, schulte, berbner, steinmetz}@kom.tu-darmstadt.de

Abstract

Business process management and business process optimization gain more and more importance in recent years. Companies have to meet customer expectations and control IT costs to stay competitive. For this, flexible and agile business processes, efficiently supported by IT systems are crucial. Web Services as an upcoming technology become more and more important establishing cross-organizational workflows. Due to a large amount of incoming requests, it is crucial to plan the workflow control. Furthermore, the resource usage has to be addressed in order to support scalability. Thus, capacity planning mechanisms are crucial for adapting a workflow to the real behavior ensuring that its execution remains feasible and SLA violations due to overload are avoided. Capacity planning requires among others a workload forecast as well as a workload modeling and analysis. Therefore, in this paper we present a holistic capacity planning and cost-effective approach for Web Service workflow parallelization and optimization.

Keywords: Web Service, Service-oriented Architecture, Capacity Planning, Business Processes, Workflows

Introduction

In recent years, the globalization and deregulation of markets forced the enterprises to react to their changing environment and therefore adapt their business processes continuously. The IT architectures within enterprises and organizations are often heterogeneous and have led to a high complexity, which is hardly manageable. A large amount of legacy systems, middleware platforms, programming languages, operating systems, and communication channels are prevailing characteristics of those architectures (Papazoglou and van den Heuvel 2000). As a crucial competitive factor, the ability to react quickly, flexibly, and efficiently to changes of the environment by adapting the business strategy to new conditions becomes more and more important (Hammer and Champy 2003). This implicates that enterprises have to be more flexible, have to plan their business processes in advance and adapt them to changing business needs. Only companies with a high flexibility to adapt to new conditions will survive in the long-term (Becker et al. 2003). A continuous business process management allows to react flexible and to manage the business process to avoid performance problems during the process execution. The business process management has to meet customer expectations, i.e. Quality of Service (QoS), and has to control the costs to stay competitive. Therefore, the key issues for a holistic process management are capacity, reliability, availability, scalability, and security (Almeida and Menascé 2002).

An architectural support for integrating internal legacy systems as well as for coupling external business partners in order to realize flexible business processes is needed. The Service-oriented Architecture (SOA) paradigm is often recommended to enable agile business processes (Papazoglou 2003). A SOA facilitates that self-contained loosely coupled services can be composed and orchestrated to cross-organizational business processes. Workflows, which are typically the automation of a business process, may use Web Services as an open standard technology that have the potential to overcome integration problems by composing and implementing business processes regardless of the underlying legacy system (Alonso et al. 2004). Web Services can be developed, deployed, and used regardless of the programming language in which the service has been

originally defined. Further, Web Services can be accessed and executed remotely through standardized Internet protocols. Usually Web Services are based on open XML standards like SOAP (W3C 2003), WSDL (Web Services Definition Language (W3C 2001)), and UDDI (Universal, Description, Discovery, and Integration), which are widely used for integration purposes within enterprises (Staab et al. 2003). Web Services as a technology based on open XML-standards have become important to realize workflows by integrating heterogeneous legacy systems and coupling IT-systems of different business partners. One of the major challenges in the area of Service-oriented Computing (SOC) is the automated and dynamic composition of Web Service workflows for the business process execution.

In our previous work we described the detailed workflow execution within one workflow (Berbner et al. 2005a, Berbner et al. 2005b) and the designed and prototypically implemented proxy-architecture Web Service Quality of Service Architectural Extension (WSQoSX), which is based on Web Service technology and enables the selection of specific Web Services at runtime. Furthermore, it facilitates the management of their Service Level Agreements (SLAs) and replanning strategies to optimize the whole workflow concerning given QoS attributes, e.g. cost, reputation, response time, and throughput. In Berbner et al. (2007) we designed and evaluated heuristics to adapt the execution plan to the actual behavior of already executed services by a dynamic service selection at runtime, ensuring that the QoS and cost requirements are still met**.**

Workflows with a high repetition rate and a high business value, e.g. claims handling, loan handling, and accounting, require a continuous workflow control including capacity planning strategies to handle all execution requests and even peaks. The capacity planning procedure concerning a complete business process management requires the following steps: understanding the environment, workload characterization, workload model validation and calibration, performance model development, performance model validation and calibration, workload forecasting, performance prediction, cost model development, cost prediction, and cost/performance analysis (Menascé and Almeida 2002). Capacity planning, in the context of workflow and business process automation, is inevitable due to the risk of poor performance and has become a major issue in recent years. Capacity planning can be described as the process of predicting when future load levels of a business process will saturate the system and determining the most cost-effective way of delaying system saturation as much as possible (Menascé et al. 1994). Considering a generic credit process, which can be decomposed into the sub processes loan request, credit assessment, servicing, and workout, represents a workflow with a high repetition rate. The high number of execution requests of the credit process has to be processed by a composed Web Service workflow. In order to ensure the complete execution of all requests a continuous capacity planning is required. It is indispensable to achieve a proactive and continuous capacity planning procedure to guarantee that all requests of a workflow can be served. Due to the necessity of a holistic workflow control for workflows with high repetition rates, we propose a capacity planning and cost-effective approach for Web Service workflows.

The rest of this research in progress paper is structured as follows. The next section shows an overview about the considered Web Service workflows. The proposed capacity planning approach for Web Service workflows including a cost-effective model supporting the capacity planning approach is described in the following section. The sustainability of the proposed approach is described in the next section with an evaluation. The paper closes with a conclusion and an outlook on future work.

Web Service workflows

In the organization theory, a business process is a consecution of activities, which creates a value to the customer. "*A business process is a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships*" (Workflow Management Coalition 1999). Workflows are typically the automation of a business process by IT-systems. "*A Workflow is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules*" (Workflow Management Coalition 1999). In order to execute a business process several Web Services WS_i (i=1,…,n) can be composed to a Web Service workflow. It is possible to select and inter-connect Web Services provided by different service providers or partners according to a business process (Zeng et al. 2004). Therefore, a business process implemented with Web Services in order to execute the basic activities of the workflow can be described as a Web Service-based workflow.

From an economical and technical aspect, the tasks within a workflow are treated as indivisible. It is not allowed to execute a task only to a specific part. Each task has to be executed completely. In the following, we consider a workflow, consisting of several specific different tasks n, which are executed sequentially and realized by Web Services. To ensure that the tasks are executed sequentially, task i (i=1,...,n) has to be executed before task i' (i'=1,...,n) if $i<i'$. For each task the workflow controller has to select the appropriate Web Service that provides the required functionality for the specific task i (i=1,...,n). In addition, the controller has to plan the whole execution of the workflow, i.e. which Web Services have to be invoked in which order. The resulting execution plan provides the sequential order and the detailed execution procedure of the workflow and

defines that the tasks are forwarded to the first Web Service WS_1. After WS_1 has executed its work package, WS_1 will invoke the next Web Service WS_2. Each Web Service will invoke the next Web Service until the workflow is completed. The last invoked Web Service gets the instruction from the execution plan to forward the results to the workflow controller. Each of these tasks has a specific but fixed response time $t_{resp,i}$ (i=1,…,n), which is specified in the SLAs with the service provider. The response time of each Web Service can be further divided into task processing time, network processing time (i.e. time consumed while traversing the protocol stacks of source, destination, and intermediate systems), and the network time itself (Repp et al. 2006). In the considered workflow the response time $t_{resp,i}$ of each Web Service WS_i can be described as:

$$t_{resp,i} = t_{task,i} + t_{stack,i} + t_{transport,i} \forall i = 1,...,n$$

Assuming that the workflow has to be instantiated in order to be executed several times due to a large amount of requests a fixed time after which the next instance execution will start has to be defined. This time is referred to as the cycle length cl, i.e. after the specific time period cl the next instance will start the execution if there is an incoming request for a workflow execution. This implies that in t_0 the execution of the first request will start and the next instance execution will start in t_1= t_0+cl. To describe the starting time of each instance execution $t_{start,e}$, we denote e as the number of request executions. The starting times for the executions are:

$$t_{start,e} = t_0 + e * cl$$

Based on the aforementioned assumption and multiple execution requests, the workflow controller has to plan the instance execution in advance to guarantee that the incoming requests can be served by the composed Web Service workflow. The resulting capacity planning problem can be seen as a simple assembly line balancing problem (Domschke et al. 1997) with the aim to parallelize workflows to guarantee that a specific number of requests can be served within a given time period. The proposed capacity planning approach to solve this problem is depicted in the following section.

Capacity planning for Web Service workflows

In this section, we present our proposed capacity planning approach to design the workflow control in advance. The aim of the approach is to guarantee that a fixed number of requests can be executed in a specific time period (Almeida and Menascé 2002).

Capacity planning approach

This approach meets the following assumptions: a fixed time between the sequential instance execution, in the SLAs is defined that the availability of the Web Services for the workflow executions are guaranteed for a specific time t_{avail}, the costs are paid per instance parallelization, and the costs per Web Service are anti proportional to the response times. As we mentioned earlier, the response time of each task is the elapsed time that occurs from invoking a Web Service until the Web Service is executed completely. After this time, the next Web Service can be invoked and the next task can be executed. Usually the response times of the Web Services are conservatively defined in the SLAs. The service provider only offers response times, which can be safely realized. Assuming that the instance execution starts in $t_{start,e}$ the instance execution completion will be in $t_{start,e}$+T. The overall execution time T of the instance is the summation of all response times of the invoked Web Services of the workflow.

$$T = \sum_{i=1}^{n} t_{resp,i}$$

The execution plan of a workflow defines the composition and the invocation of the Web Services. In order to analyze, whether the workflow is able to serve all incoming requests, the capacity respective the number of parallel executions of the workflow and the workload of the process have to be analyzed. In order to serve all incoming execution requests the workflow has to be available a specific period of time t_{avail} per day and has a specific number of minimal requests r per day. An upper bound for the cycle length cl_{max} of the workflow (the elapsed time of starting a workflow until the workflow is executed completely) can be calculated by the division of the time, where the workflow has to be available and the number of requests in this time period. The maximum cycle length is defined by:

$$cl_{\max} = \frac{t_{avail}}{r}$$

The cycle length cl_{max} is the maximum length to ensure that the composed workflow can serve all incoming requests of a workflow execution, i.e. after the time cl_{max} the next instance has to start to guarantee that all incoming request can be served. In addition to the maximum cycle length cl_{max}, we can specify a lower bound for the number of instances w_{min} that have to be parallelized to guarantee that all requests r can be executed by the specific workflow. The calculation of the number of parallel instances $w_{\min}$ can be done by dividing the overall execution time T of the instance by the maximum cycle length cl_{max}. The formula for w_{min} results to:

$$w_{\min} = \left\lceil \frac{T}{cl_{\max}} \right\rceil$$

If the overall execution time T, which is restricted by the response times of the invoked Web Services, exceeds the maximum cycle length cl_{max}, instances have to be parallelized to ensure that all requests can be served. In this scenario, it is important to consider that $w_{\min}$ can only accept whole numbers due to the indivisibility of instances. It is not feasible to execute a Web Service respective a task within one instance execution in part.
The introduced formulas define an upper bound for the cycle length $cl_{\max}$, the overall response time T of one instance execution and the number of parallelized instances $w_{\min}$. A measure to quantify the utilization ratio of the composed workflow based on the cycle length and workflow parallelization is the level of capacity of the workflow. The level of capacity LC, where cl denotes the actual used cycle length can be described as:

$$LC = \frac{T}{w * cl}$$

Each parallelization of the instances leads to higher costs to execute all incoming execution requests, because the service provider has to provide more services at the same time. He also has to increase his capacity to be able to offer parallel services and is therefore faced with higher costs, which are passed on the service consumer. The aim of the workflow controller during the capacity planning process is to maximize the capacity of the designed workflow. Thus, the level of capacity has to be optimized for each instance parallelization for an optimal usage of the workflow. In order to achieve a high level of capacity, the workflow controller has to analyze all minimum and maximum cycle lengths within one instance parallelization. The minimum and maximum cycle length for the parallelization of instances $cl_{min,w}$ and $cl_{max,w}$ can be derived as follows:

$$cl_{\min,w} = \frac{T}{w} \forall w = 1,...,m$$

$$cl_{\max,w} = \lim_{p \to w-1} \frac{T}{p} \forall w = 1,...,m$$

Assuming that the overall response time T of an instance is fixed and the workflow controller has made a decision for the number of parallelization, an optimal level of capacity can be achieved by selecting a small cycle length. Due to the indivisibility of Web Services and the minimum number of request r in the available time period t_{avail}, the chosen minimum cycle length has to be smaller than cl_{max}. Concerning the formula for LC an optimization will be achieved by using the minimum cycle length $cl_{min,w}$. This implies, that after the workflow controller has made a decision how many instances he wants to parallelize, he has to choose the minimum cycle length $cl_{min,w}$ of the corresponding parallelization w to achieve an optimum for the level of capacity.

Obviously, for each number of parallelization we can determine the optimal level of capacity. The challenge for the workflow controller is to minimize the execution time for the workflow and to minimize the costs. Execution time in this context means that the smaller the cycle length cl, the faster the workflow controller can execute the requests. Without parallelization, the requestors have to wait a long period of time until their requests will be executed. The minimization of execution time yields to a high number of parallelization and therefore higher costs. To handle the tradeoff between execution time, i.e. cycle length, and the resulting costs the next subsection describes our proposed cost-effective model.

Cost-effective model for the capacity planning approach

In the following analysis, we assume that the workflow controller has to pay fixed costs per service for the composition of the workflow, i.e. each invoked Web Service in the workflow has specific costs c_i (i=1,…,n) which are specified in the Service Level Agreements with the service provider. Therefore, the overall costs C for the provision of one workflow are the summation of all costs c_i (i=1,…,n). The overall costs for the workflow controller depend on the number of instance parallelization w, because a parallelization of instances leads to a multiple use of services within a fixed time period. The service provider will demand higher costs if he has to offer multiple services to realize instance parallelization. We denote the costs to serve all incoming request depending on the number of parallelization w with C_w.

$$C = \sum_{i=1}^{n} c_i$$

$$C_w = w * C$$

The workflow controller is faced with a tradeoff between cost reduction and achieving small cycle lengths. He has to make a decision how many instances at reasonable costs he wants to parallelize to obtain a passable cycle length. A purely cost minimizing approach would be to choose the derived maximum cycle length cl_{max} to achieve a small number of instance parallelization. In fact, the requestor should wait a long time until all requests are processed. Thus, the workflow controller also has to be concerned about the cycle lengths and the instance execution times to enable small waiting times for the request. The controller has to make a decision which cycle length is acceptable at which costs, which is dependent on his time and costs preferences. In addition, he can renegotiate with the service provider to offer services, with smaller response times. The workflow controller has to decide how many instances he wants to parallelize based on increasing costs per parallelization. For a further cycle length and cost reduction, he has to negotiate with the Web Service provider to change the SLAs to smaller response times for each Web Service execution. After the negotiation of smaller response times, this time reduction leads to a decrease in the overall response time T, a decrease in the minimum instance parallelization and a cost reduction for the workflow executions C_w, even it might be that the costs per instance C increase due to the new SLA negotiation. The correlation between a reduction of the overall response time Δx (in percent) and the impact on the cycle length by a fixed number of parallelization is:

$$cl_{\min,w} = \frac{T}{w} - \frac{\Delta x * T}{w} = (1 - \Delta x) * \frac{T}{w},$$

This implies that if the overall response time of one instance execution is reduced by Δt %, the minimum cycle length $cl_{min,w}$ will be reduced by the same percentage. The reduction in instance parallelization with a fixed minimum cycle lengths $cl_{min,f}$ before the SLA renegotiation can be described as:

$$w = \left\lceil (1 - \Delta x) * \frac{T}{cl_{\min,f}} \right\rceil$$

With the identified approach, the workflow controller is able to analyze the instance parallelization due to the overall response time, the requests in a given time period, the cycle length and the overall costs to serve all execution requests. The workflow controller is faced with the tradeoff of response time reduction and an increase in costs per service. Depending on the reduction of response time and the increase in costs per service, the workflow controller is able to measure the instance parallelization and the overall execution costs based on his costs and execution time preferences. In the next section, we evaluate the capacity and cost planning approach with an example.

Evaluation

In order to evaluate our approach we consider a process with a high repetition rate and a high business value. In the previous section, we depicted formulas how to calculate the maximum cycle length cl_{max}, the level of capacity of the considered workflow LC and the overall costs for the instance parallelization C_w. We assume that the business process can be decomposed into n=*14* different tasks as a whole, i.e. *14* different Web Services which fulfill the functional requirements. Dependent of the analyzed process, the number of different tasks n may vary. The response times of the invoked Web Services, which ful-

fill the required functionality for each specific task, are depicted in Table 1. It can be seen that the execution of one instance needs at least *T=1,1* sec in this example.

Table 1. Response time of the invoked Web Services

	WS_1	WS_2	WS_3	WS_4	WS_5	WS_6	WS_7	WS_8	WS_9	WS_{10}	WS_{11}	WS_{12}	WS_{13}	WS_{14}	T
$t_{resp,i}$ (sec.)	0,1	0,05	0,07	0,08	0,08	0,07	0,08	0,08	0,05	0,08	0,09	0,09	0,09	0,09	1,1

We assume that the business process has a minimum number of requests *r=100.000* per day and the time where the process has to be available is t_{avail}*=24h*. Calculating the maximum cycle length which is at least necessary to serve all incoming requests with the above mentioned formulas leads to cl_{max}*=0,864* sec. Obviously cl_{max} is smaller than the overall response time *T* for one instance execution. This implies that if the workflow controller would execute one request after another it would not be possible to serve all incoming request during the availability time t_{avail}. Thus, the instance execution has to be parallelized to guarantee that all requests can be served within t_{avail}. Thus, for this scenario w_{min}*=2*, i.e. the workflow controller has to operate at least two parallel workflows.

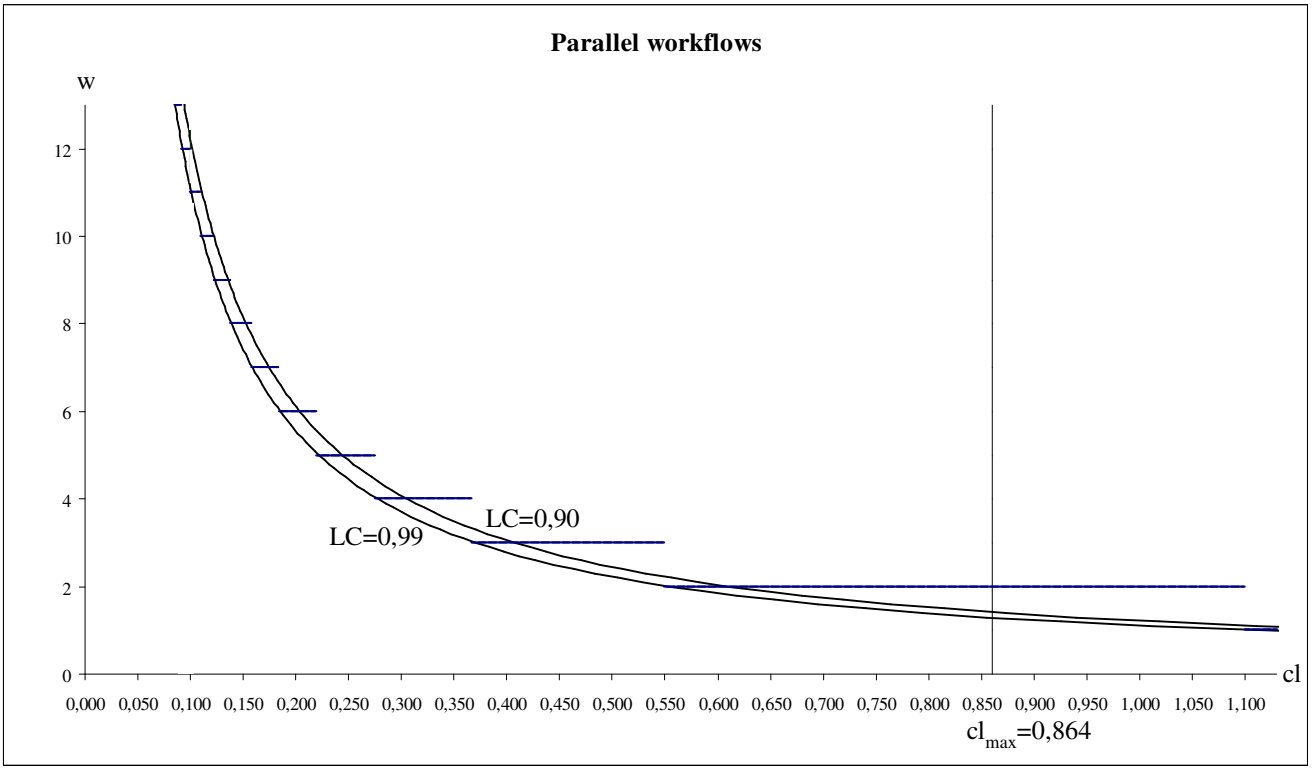

Figure 1. Parallel workflow execution

Figure 1 depicts the correlation between cl_{max}, w_{min}, and the level of capacity *LC* of the workflow. In this example the level of capacity is set to *LC=0,9*, which implies that the workflow has a level of capacity of *90%*. The workflow has still the potential to increase its efficiency. In order to optimize the workflow efficiency with respect to the level of capacity, the isoquant of LC has to be shifted until the isoquant reaches the minimum number of instance parallelization. This implies that graphically the resulting isoquant will touch the minimum cycle lengths for each instance parallelization as mentioned earlier. Both the minimum and the maximum cycle lengths of the introduced scenario are depicted in Table 2.

Table 2. Minimum and maximum cycle lengths of the workflow parallelization

w	1	2	3	4	5	6	7	8	9	10	11	12
$cl_{max,w}$ (sec.)	∞	1,099	0,549	0,366	0,274	0,219	0,183	0,157	0,137	0,122	0,109	0,099
$cl_{min,w}$ (sec.)	1,100	0,550	0,367	0,275	0,220	0,184	0,158	0,138	0,123	0,110	0,100	0,092

As shown in Figure 1 the workflow controller may parallelize at least *two* workflows to serve all incoming requests. In order to make a decision of how many instances have to be parallelized Table 3 shows the resulting operating costs for the number of parallel instances (assuming the costs for one parallelized instance is C=0,15 $) depending on the minimum cycle lengths. The coherence between workflow parallelization and costs is linear.
Based on these characteristics the workflow controller has to analyze the execution time preferences as well as his cost preferences. Obviously, it is not trivial to make the decision how many instances have to be parallelized, because of the different preference structures of the workflow controller. Assuming that the main objective of the workflow controller is to minimize the cycle length and realize a fast execution of all requests, he can parallelize as many instances as he wants which also leads to higher costs.
In contrast, if the workflow controller has the objective to minimize his costs, he has to apply at least $w=2$ workflows due to the restriction of $cl_{max}=0,864$. According to the implementation of $w=2$ workflows the workflow controller has further the possibility to realize quite smaller cycle lengths up to $cl_{min,2}=0,55$ sec. to execute all requests $r=100.000$.

Table 3. Parallelization costs

w_{min}	1	2	3	4	5	6	7	8	9	10	11	12
$cl_{min,w}$ (sec.)	1,100	0,550	0,367	0,275	0,220	0,184	0,158	0,138	0,123	0,110	0,100	0,092
C_w ($)	0,15	0,30	0,45	0,60	0,75	0,90	1,05	1,20	1,35	1,50	1,65	1,80

Assuming that the workflow controller is restricted by a fixed budget for the whole workflow execution, he has to choose a number of instance parallelization at reasonable costs. Afterwards he can calculate the resulting cycle length by choosing the minim cycle length $cl_{min,w}$, dependent on his chosen w. After he made his decision, he has the possibility to renegotiate with the service provider to decrease the response times for the invoked Web Services. This may lead to a small increase in costs per instance C, but the fact that smaller response times allow a smaller number of instance parallelization, which has a positive effect on the overall costs C_w , leads to cost savings for the workflow controller in the long term.

Conclusion and outlook

In this paper, we propose a capacity planning mechanism for Web Service workflows to ensure that all incoming requests for a workflow in a given timeframe can be served. We also propose a cost model to minimize the workflow costs for parallel executions. Due to the preference structure and the resulting costs for the parallelization, the workflow controller is able to parallelize instance executions at reasonable costs.

Our further research aims at extending our approach to other cost models and preference structures of the workflow controller. We will enhance the capacity planning approach considering other resource planning problems, which may also occur during a workflow execution. Further, we will focus on other workflows with a high repetition rate and a high business value, to simulate and further evaluate the approach proposed in this paper. For this, we start to implement a simulation engine to analyze the trade-off between costs, the reduction in response time and instance parallelization. We will also evaluate our approach in business scenarios within the E-Finance industry.

Acknowledgments

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

References

1. Almeida, V. A. F., and Menascé, D. A. “Capacity Planning: An Essential Tool for Managing Web Services,” IT Professional (4:4), 2002, pp. 33-38.
2. Alonso, G., Casati, F., Kuno, H., and Machiraju, V. Web Services. Concepts, Architectures and Applications, Berlin, Germany: Springer, 2004.

3. Becker, J., Kugeler, M., and Rosemann, M. Process Management. A Guide for the Design of Business Processes. Berlin. Germany: Springer, 2004.
4. Berbner, R., Grollius, T., Repp, N., Heckmann, O., Ortner, E., and Steinmetz, R. An approach for the Management of Service-oriented Architecture (SoA) based Application Systems, Enterprise Modelling and Information Systems Architectures (EMISA 2005), Klagenfurt, Austria, 2005.
5. Berbner, R., Heckmann, O., and Steinmetz, R. An Architecture for a QoS driven composition of Web Service based Workflows, Networking and Electronic Commerce Research Conference (NAEC 2005), Riva del Garda, Italy, 2005.
6. Berbner, R., Spahn, M., Repp, N., Heckmann, O., and Steinmetz, R. Dynamic Replanning of Web Service Workflows, IEEE International Conference on Digital Ecosystems and Technologies 2007 (IEEE DEST 2007), Cairns, Australia, 2007.
7. Domschke, W., Scholl, A., and Voß, S. Produktionsplanung – Ablauforganisatorische Aspekte, Berlin, Germany: Springer, 1997.
8. Hammer, M., and Champy, J. Reengineering the Corporation: A Manifesto for Business Revolution, New York, USA: HarperBusiness, 2003.
9. Menascé, D. A., and Almeida, V. A. F. Capacity Planning for Web Services: Metrics, Models, and Methods, Upper Saddle River, New Jersey: Prentice Hall, 2002.
10. Menascé, D. A., Almeida, V. A. F., and Dowdy, L. D. Capacity Planning and Performance Modeling: From Mainframe to Client-Server Systems, Upper Saddle River, New Jersey: Prentice Hall, 1994.
11. Papazoglou, M. P. Service-Oriented Computing: Concepts, Characteristics and Directions, 4th International Conference on Web Information Systems Engineering (WISE 2003), Rome, Italy, 2003, pp. 3-12.
12. Papazoglou, M. P., and van den Heuvel, W. J. "Leveraging legacy assets," in: Papazoglou, M. P., Spaccapietra, S. and Tari, Z. (Eds.), "Advances in Object-Oriented Modeling," MIT Press, Cambridge, MA, 2000, pp.131-160.
13. Repp, N., Berbner, R., Heckmann, O., and Steinmetz, R. A Cross-Layer Approach to Performance Monitoring of Web Services, ECOWS 2006 Workshop on Emerging Web Services Technology, IEEE, Zurich, Switzerland, 2006: ISSN 1613-0073, vol. 234, http://ceur-ws.org/Vol-234/paper2.pdf.
14. Staab, S., Benjamins, R., Bussler, C., Fensel, D., Gannon, D., Maedche, A., Sheth, A., and van der Aalst, W. M. P. "Web services: Been there, Done that?" IEEE Intelligent Systems, Trends & Controversies (8:1), 2003, pp. 72-85.
15. W3C, SOAP Version 1.2, W3C Recommendation, http://www.w3.org/TR/soap12/, 2003.
16. W3C, Web Services Description Language (WSDL) 1.1, W3C Note, http://www.w3.org/TR/wsdl, 2001.
17. Workflow Management Coalition, "Workflow Management Coalition – Terminology & Glossary," Technical Report WFMC-TC-1011, Hampshire, United Kingdom, 1999.
18. Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., and Chang, H. "QoS-aware Middleware for Web Service composition," IEEE Transactions on Software Engineering (30:5), 2004, pp. 311-328.