A Price of Service in a Compositional SOA Framework

Kees van Hee[1], Natalia Sidorova[1], Christian Stahl[2], and Eric Verbeek[1]

[1] Department of Mathematics and Computer Science,
Eindhoven University of Technology
PO Box 513, NL-5600 MB Eindhoven, The Netherlands.
{k.m.v.hee,n.sidorova,h.m.w.verbeek}@tue.nl
[2] Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin
stahl@informatik.hu-berlin.de

Abstract. In this paper we propose a framework for SOA covering such important features as proper termination (soundness) and correct correlation of tasks. Within this framework, we define a method for the calculation of the price of services. Our framework is compositional in the sense that composing a system from subsystems that meet our correctness requirements we obtain a system that still meets these requirements.

1 Introduction

In the Service Oriented Architecture (SOA) paradigm, a system is a network of components. Each component can be a service provider and a service client at the same time. Through its *sell-side interface*, the service offers and delivers services to its clients, and through its *buy-side interface* it may request services from suppliers. Both clients and providers may be other components.

The service a component may deliver is defined by a *workflow*, which is also called the *orchestration* of the service. This workflow consists of a partially ordered set of *tasks*. Each task may invoke a new service instance of another component. The interaction between the components is depending on the communication protocols and is called the *choreography*.

Since SOA-based systems are open in the sense that components may enter and may leave the network, we have to take into account the possibility that a service breaks down during service delivery, i.e. a service may fail. For a SOA there is an obvious *correctness criterion*: each service should always return a result to its client: either it delivers the requested items or it returns a failure signal. After the result is reported, all service activities are terminated. We call this property the *proper completion* of a service. We may require a system to have this property independent of the number of active components in the system.

One more essential but still not elaborated feature of the SOA paradigm is the similarity with the *supply chain paradigm*. According to this paradigm components are more or less *autonomous agents* that deliver their services in

return for a reimbursement and they pay their suppliers for the services they buy in order to deliver their own service. Only services with an (expected) positive added value, i.e. services that on average receive more money than they spend, will survive. So besides the correctness criteria there is a *performance criterion* that says that components should only provide services if their expected added value is positive. We call this property *profitability*.

These SOA principles are described by many authors and the concepts start to converge to common understanding. There exist languages such as BPEL [3] and WSDL [7] that allow us to define orchestration and to some end choreography. However there is, to our best knowledge, no example of a components framework that has covered all features described above in a consistent way. As a "proof of concept" we construct such a components framework in this paper.

We make some simplifying assumptions on the protocols for sell and buy side and on the stochastic process of finding suitable suppliers. The framework can easily be extended to other, more realistic, assumptions. We give sufficient conditions for a SOA-based system to satisfy both the proper completion and the profitability criterion. Moreover, a system composed within our framework from components that meet these two requirements will also meet these requirements, i.e. our framework is *compositional* with respect to them.

In Section 2 we introduce some basic notions. In Section 3 we introduce the components framework and show that all service instances have the proper completion property. In Section 4 we introduce the economical aspects of services: the cost parameters and the probabilities for choices in the orchestration and give sufficient conditions for components to be profitable. We conclude in Section 5 with a discussion of possible extensions, a review of related works and future work.

2 Preliminaries

In this section we introduce the basic definitions we need in the rest of the paper.

Petri nets A Petri net [10, 12] is a bipartite graph whose nodes are places and transitions. For the sake of simplicity, we assume that a Petri net contains at least one place.

Definition 1 (Petri net). *A Petri net is a tuple $N = (P, T, F)$ where P is a (non-empty finite) set of places, T is a set of transitions, $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs.*

To define the state of a net, we associate to every place of a Petri net $N = (P, T, F)$ a (non-negative) counter. The values of the counters of all places of the net form a *marking* (state) of the net. A marking $M \in \mathcal{M}(N)$ can be interpreted as a vector, function, or multiset over the set P of places. A marking $M \in \mathcal{M}(N)$ is visualized by putting $M(p)$ *tokens* (black dots) into every place p.

Firings of transitions may change the state of a net. A transition t may fire in marking M when it is *enabled* in M, i.e. every its input place p (a place

such that $(p,t) \in F$) contains at least one token. If an enabled transition fires, it removes one token from every its input place and adds a token to every its output place (i.e. a place p such that $(t,p) \in F$). We denote by $M \xrightarrow{t} M'$.

A Petri net $N = (P,T,F)$ together with a marking M is called a marked Petri net, denoted (N,M). A marked Petri net induces a state space, where every state corresponds to a marking that can be reaches as a result of some firing sequence. This set of reachable markings is called the reachability set of the marked Petri net (N,M) and is denoted $N[M\rangle$.

Worfklow nets A workflow net (WF-net) [1] is a Petri net that is specifically tailored towards modelling workflow processes. Typically, a workflow has a well-defined point of entry (where new cases start) and a well-defined point of exit (where handled cases end). Furthermore, every activity in a workflow typically forwards the case from the point of entry to the point of exit. These three requirements are reflected in workflow nets.

Definition 2 (WF-net). *A workflow net is a Petri net $N = (P,T,F)$ such that (1) there exists exactly one input place i, i.e. ${}^\bullet i = \emptyset$, (2) there exists exactly one output place o, i.e. $o^\bullet = \emptyset$, and (3) every place and transition lies on some path from i to o.*

An important correctness property of workflows is soundness [1], which comprises the requirements that for every case the point of exit can be reached, and that when this point is reached no work is being left behind, and moreover for every activity there are some cases possible for which the activity can be executed.

Definition 3 (soundness). *A workflow net $N = (P,T,F)$ is called* sound iff *(1) for every $M \in N[i\rangle$: $o \in N[M\rangle$, and (2) for every $t \in T$ there exists a $M \in N[i\rangle$ such that M enables t.*

Soundness can be automatically checked by a number of Petri net tools, like the tool Woflan [13, 14].

3 Components framework without values

In the SOA domain, Petri nets are used to model components that communicate with each other through an interface, which is typically modelled by places [9]. For this reason, we distinguish interface places from other places. An interface place can either be an input place or an output place. An input place has no input arcs, whereas an output place has no output arcs. Note that, as a result, a component is not allowed to communicate with itself.

As mentioned in the introduction, we can distinguish between sell-side components and buy-side components. Typically, when some service wants to use another service, that is, if the former consumes a service that is being provided by the latter, the former first requests a quote from the latter. Based on the offer

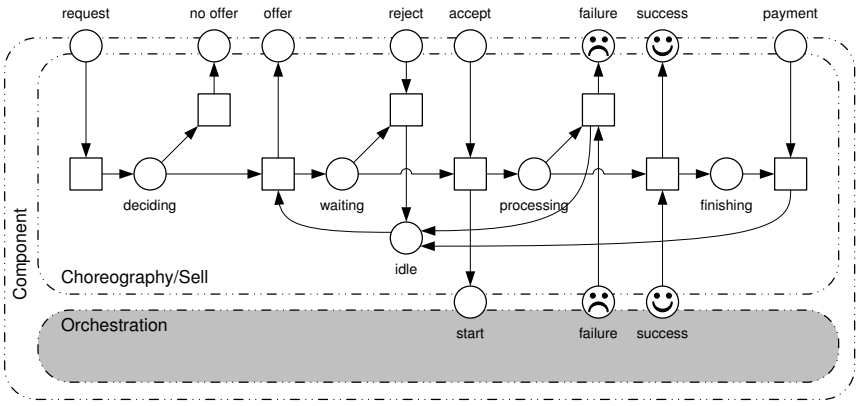

Fig. 1. The sell-side of a component

from the provider (which is optional, as the provider may decide not to offer the service), the consumer decides to either accept or reject the offer. If the offer is accepted, the provider actually provides the service, which might either succeed or fail. This result is communicated to the consumer, which pays the provider if the result was a success (no-cure-no-pay).

To provide the service, the provider might have to consume third-party services in some order. Clearly, the provider needs to *orchestrate* these third party services on-the-fly to achieve its goal. In contrast, the negotiation between the provider and the customer is more of a *choreographed* nature.

Choreography The choreography in the framework consists of the sell-sides of the components and the buy-sides of the tasks. Figure 1 visualizes the sell-side of a component, whereas Figure 2 visualizes the buy side of a task. As usual, circles represent places and squares represent transitions. For the ease of reference, sad smileys have been used for the failure places and happy smileys for the success places.

Underlying assumption for the sell-side of a component is that a component can handle a predefined number of requests simultaneously. This predefined number corresponds to the number of tokens which are initially put into the place idle. Thus, if the maximum number of requests is being handled, then no offer can be made for the next request.

The buy-side of a task includes a place that contains a number of *prospects*. A prospect corresponds to a third-party service that can actually perform the task. If the task is started, an undefined number of prospects is queried for a quote. If a prospect denies the request (no offer), the prospect is done. Otherwise, the quote can either be accepted or rejected by the buy-side of the task.

Orchestration The service a component performs can be a *simple task*, like retrieving some information from an underlying database, but it can also be a

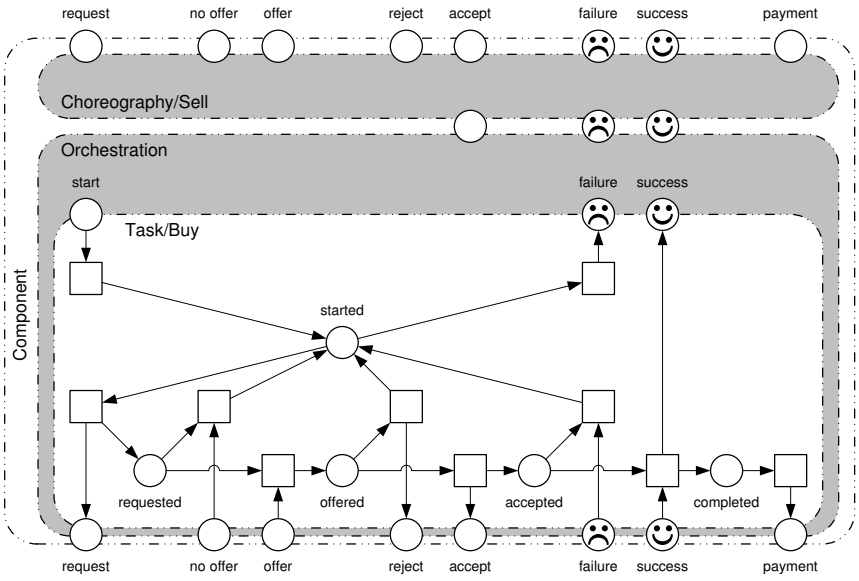

Fig. 2. A task in the buy-side of a component

compound task that needs to orchestrate a number of (sub)tasks. In principle, such an orchestration can be arbitrarily complex, but in this paper we consider four types of operations to construct compound tasks:

1. *sequence*, i.e. performing a number of tasks in a given order,
2. *parallel composition*, i.e. performing a number of tasks simultaneously,
3. *choice*, i.e. performing one task chosen from several tasks, based on some decision, and
4. *while*, i.e. performing a task as long as some condition holds.

Nevertheless, we would like to stress that the framework can be extended with additional orchestration types if needed (as long as soundness is guaranteed, see next section). Reason for restricting to this set of operations in this paper, is that these four types are sufficient to explain the matters at hand, whereas additional types might only distract the reader.

Figures 3–6 visualize these four basic orchestration types. Again, we use the sad smileys for the failure places and the happy smileys for the success places, and we use grey boxes to visualize the orchestrated tasks. For the sake of simplicity, we used only two tasks for the sequence, parallel composition, and choice, but it is not hard to see that this scheme can be extended to any number.

We provide a brief explanation for the parallel orchestration: This orchestration fails if any of the tasks fails, and succeeds if all the tasks succeed. An alternative would be to use 2^n transitions (where n is the number of tasks) to handle this orchestration scheme, but such a scheme scales badly leading to the

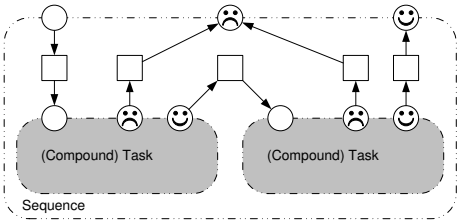

Fig. 3. A sequence of tasks

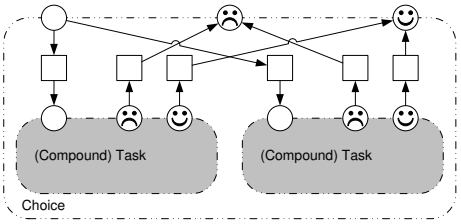

Fig. 4. A choice between tasks

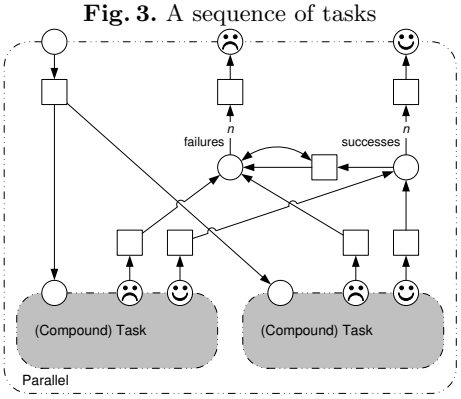

Fig. 5. A number of tasks in parallel

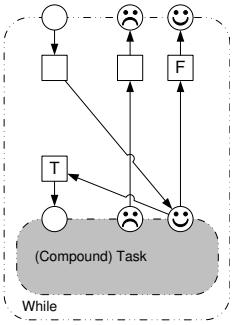

Fig. 6. A while construct

net explosion. Therefore, we use a scheme that requires only $2n + 3$ transitions and 2 additional places: failures and successes. These places hold the failure and success messages (tokens). If n successes have been signalled, and hence no failures, the orchestration may signal success. If some failures have been signalled, then we allow the signalled successes to be converted into failures. After all successes have been converted this way, the orchestration can signal its failure.

For the *while* case, if the specified condition holds, then the T transition can fire, otherwise the F transition can fire.

Example Figure 7 shows an example service using the component framework, containing one choreography component (the sell side of the service), three orchestration components (which are compound tasks), and four (simple) task components (the buy side of the service). The choreography component and the task components are fixed (see Figures 1 and 2), but the orchestration components are not (so they are not displayed). Using the orchestration components (see Figures 3 to 6) we can build a complex orchestration hierarchy.

Orchestration soundness Clearly, any orchestration should lead to either a success or a failure. For this, we can simply use the soundness property, using the scheme as visualized by Figure 8. As a result, an orchestration is called sound if

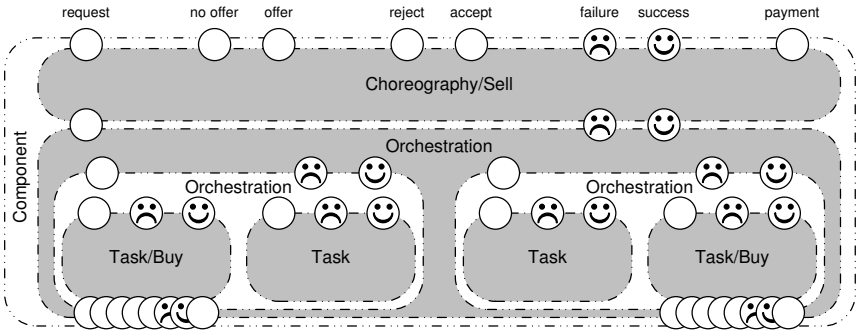

Fig. 7. An example service

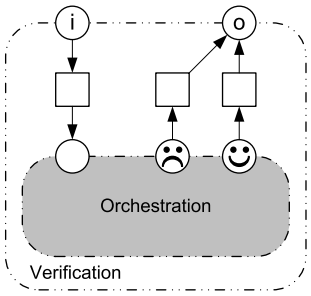

Fig. 8. Checking orchestration soundness

and only if the orchestration extended with the scheme visualized by Figure 8 is sound.

However, we argue that we only need to check soundness on simple tasks, as any orchestration is sound if and only if its tasks are sound. This is straightforward to check using only Figures 3 to 6. In case the framework is to be extended by some new orchestration types, we need to check this requirement (the orchestration is sound if and only if the tasks are sound).

A possible implementation in BPEL In order to show that our framework is practically feasible in a SOA setting, we show how the sell side of a component could be implemented in BPEL [3]. The resulting BPEL model is abstract, as it only contains the control-flow perspective. To keep the listing as short as possible, we have removed lines which only contain end-tags. After all, the indentation of the listing contains sufficient information to have these end-tags restored. The BPEL process mimics the behavior of the Petri net fragment shown in Fig. 1. Note that we need a way to keep track of the number of simultaneously running instances for this process, and that we might have to decide whether or not we want to make an offer.

```
1 <process name=“Sell side of the component’’>
```

```
<partnerLinks>
  <partnerLink name="client" partnerLinkType="any"
               myRole="component" partnerRole="client"/>
  <partnerLink name="task" partnerLinkType="any"
               myRole="component" partnerRole="task"/>
<sequence>
  <receive partnerLink="client" operation="request"
           createInstance="yes"/>
  <if>
    <condition>number of instances &lt; maximum and want to offer</condition>
    <sequence>
      <invoke partnerLink="client" operation="offer"/>
      <pick>
        <onMessage partnerLink="client" operation="reject">
          <empty/>
        <onMessage partnerLink="client" operation="accept">
          <sequence>
            <invoke partnerLink="task" operation="start"/>
            <pick>
              <onMessage partnerLink="task" operation="failure">
                <invoke partnerLink="client" operation="failure"/>
              <onMessage partnerLink="task" operation="success">
                <sequence>
                  <invoke partnerLink="client" operation="success"/>
                  <receive partnerLink="client" operation="payment"/>
    <else>
      <invoke partnerLink="client" operation="no offer"/>
```

4 Value-based services

In this section we will extend the model by assigning a cost to each outsourced task. Each task has a maximal acceptable price, which means that if no service provider is willing or able to perform the task for an amount of money within this price limit, we consider this as a failure of the task. Note that the outsourcing of a task may also fail due to problems with the service provider, but that is a different and independent cause. Remember that a service provider gets only paid after delivering its service. First, we will present the cost model for the outsourcing of one task. Afterwards we will compute the total expected cost of the an offered service, i.e. the expected cost of all tasks in one orchestration. The price we ask for a service should be at least the expected cost of the execution of the service in order to be profitable.

4.1 Expected cost of a simple task

Each task needs one type of service. For each service we request, we suppose to know a discrete probability distribution p such that the lowest price X we have to pay equals k with probability p_k, i.e. $\mathbb{P}[X = k] = p_k$. For each task there is

a maximal price we are prepared to pay: m. So the expected cost of a task is $\sum_{i=1}^{m} i \cdot p_i$.

The probability that the outsourcing of a task succeeds depends not only on the deal making with a service provider but also on the successful execution of the service provider. Each service provider has a probability of successful delivery of the service. Let this probability be q and we assume that failure is an independent event. Then the probability s of successful outsourcing is $s := q \cdot \sum_{i=1}^{m} p_i$ and the probability that the outsourcing fails is $1 - s$. The expected cost C of a task is $C := q \cdot \sum_{i=1}^{m} i \cdot p_i$ since we only have to pay for a service if it has completed successfully. Note that the conditional expected cost, given that the task is successful, is C/s which equals $(\sum_{i=1}^{m} i \cdot p_i)/(\sum_{i=1}^{m} p_i)$. With these data we can make the model for the whole service.

A question to be answered is whether this is a realistic model of the cost of a task. In order to answer it, we consider a more detailed model of the service brokerage. We assume there are many service providers that can deliver the requested service, and they all have their own price. So there are a_k providers that will deliver the service for k where $k = 1, 2, 3, \ldots$. We assume that the sequence $a = (a_1, a_2, a_3, \ldots)$ is unbounded. So we assume that if the price is high enough there is always somebody who will provide the service. The cumulative number of providers that is prepared to offer the service for price not higher than k is $A_k = \sum_{i=1}^{k} a_i$. At some point in time, when our task will request the service, there are N other tasks before us that require the service simultaneously. Here N is random variable. So the probability that we will get the cheapest offer for price k is:

$$\mathbb{P}_k = \mathbb{P}[A_{k-1} \leq N < A_k] = \sum_{i=A_{k-1}}^{A_k - 1} \mathbb{P}[N = i]. \tag{1}$$

We assume that the a-sequence can be looked up, for example in the UDDI, and that the probabilities can be estimated from the log files of the UDDI.

4.2 Expected cost of a compound task

As derived above, a simple task t has expected cost $C(t)$, a probability of successful termination $s(t)$ and a probability of failure $1 - s(t)$. Next we will consider the expected cost of a compound task. We will do this using the principle of structural induction, where we consider the four construction rules of Section 3. Consider two tasks a and b with expected cost $C(a)$ and $C(b)$, respectively and with success probabilities $s(a)$ and $s(b)$, respectively. We denote the sequential coupling of a and b by $a.b$, the parallel coupling by $a||b$, the choice coupling by $a + b$ and the while iteration of task a by a^*. We may assume these task are either be simple or compound.

Sequence The expected cost of the compound task is

$$C(a.b) = C(a) + s(a) \cdot C(b) \tag{2}$$

and the success probability is

$$s(a.b) = s(a) \cdot s(b). \tag{3}$$

To verify this note that if task a has finished successful then we have to pay the service provider and if it is not successful, then we do not start task b.

Parallel The expected cost of the compound task is

$$C(a||b) = C(a) + C(b) \tag{4}$$

and the success probability is

$$s(a||b) = s(a) \cdot s(b). \tag{5}$$

Note that we have to pay a service provider if the task has successful terminated independent of the other task, but the whole task is only successful if both tasks were successful.

Choice Assume that task a is chosen with probability α and task b with probability β. Then the expected cost of the compound task is

$$C(a+b) = \frac{\alpha}{\alpha+\beta} \cdot C(a) + \frac{\beta}{\alpha+\beta} \cdot C(b) \tag{6}$$

and the probability of success is

$$s(a+b) = \frac{\alpha}{\alpha+\beta} \cdot s(a) + \frac{\beta}{\alpha+\beta} \cdot s(b). \tag{7}$$

While Here we have to consider the probability of repetition. Let the probability of action T in Figure 6 be α and the probability of action F be $1-\alpha$. The expected cost is

$$C(a^*) = \alpha \cdot C(a) + \alpha \cdot (\alpha \cdot s(a)) \cdot C(a) + \alpha \cdot (\alpha \cdot s(a))^2 . C(a) + \ldots = \frac{\alpha \cdot C(a)}{1 - \alpha \cdot s(a)} \tag{8}$$

and the success probability is

$$s(a^*) = (1-\alpha) + (1-\alpha) \cdot (\alpha \cdot s(a)) + (1-\alpha) \cdot (\alpha \cdot s(a))^2 + \ldots = \frac{1-\alpha}{1 - \alpha \cdot s(a)}. \tag{9}$$

Note that each cycle of the task has expected cost $C(a)$, independent if it is a success or a failure.

So if we start with simple tasks, the four rules give us the expected cost and success probabilities of the compound tasks. So with structural induction we can define these characteristics for all compound tasks. Important for this *associativity* of the first three construction rules:

$$C((a.b).c) = C(a) + s(a) \cdot C(b) + s(a) \cdot s(b) \cdot C(c) = C(a \cdot (b \cdot c)) \tag{10}$$

$$s((a.b).c) = \quad s(a) \cdot s(b) \cdot s(c) \quad = s(a \cdot (b \cdot c)) \tag{11}$$

$$C((a||b)||c) = \quad C(a) + C(b) + C(c) \quad = C(a||(b||c)) \tag{12}$$
$$s((a||b)||c) = \quad s(a) \cdot s(b) \cdot s(c) \quad = s(a||(b||c)) \tag{13}$$

$$C((a+b)+c) = \quad \tfrac{\alpha+\beta}{\alpha+\beta+\gamma} \cdot (\tfrac{\alpha}{\alpha+\beta} \cdot C(a) + \tfrac{\beta}{\alpha+\beta} \cdot C(b)) + \tfrac{\gamma}{\alpha+\beta+\gamma} \cdot C(c) \quad = C(a+(b+c)) \tag{14}$$
$$s(a+(b+c)) \quad = \quad s(a+(b+c)) \tag{15}$$

where we have for each task probabilities α, β and γ.

So a service can only be profitable if the *price* P that has to be asked to the client satisfies $s \cdot P > C$ where C is the expected cost of the compound task of the service and s is the success probability.

5 Conclusion

In our previous work [2], we have developed a SOA-based architecture framework which is similar to the service component architecture [4]. In this paper, we extended this work by a component framework which allows to check the soundness property compositionally. Furthermore, our model takes the price of a service into account.

Nonfunctional properties, also known as quality of service (QoS), are of increasing importance when designing a service oriented architecture. In this paper, we restricted our approach to profitability; that is, to cost. Cardoso et al. [6] present a QoS model for time, reliability, and cost of workflows. Each task has a QoS attribute. Based on these attributes, the cost of the overall workflow can be computed using the METEOR workflow system. In [15], Zeng et al. present a framework for QoS-aware service selection. Price is one of the nonfunctional properties which are taken into account. Paoli et al. [11] address the problem of designing a composed system that has to guarantee certain quality criteria such as security, completion time, and also costs. It is shown how these criteria can be computed on the structure of a service. To this end, quality evaluation rules (similar those in [6]) for sequence, parallel, switch, and loop are proposed. To summarize, all three approaches cover a broader spectrum of QoS criteria than costs. However, probabilities for successful termination and for the price calculation of tasks and services are not considered. In [5], the authors present a general framework for the evaluation of the financial consequences of outsourcing, while in [8] the authors investigate pricing mechanisms appropriate for web services. This is, however, far beyond the scope of this paper.

For the future work we plan to extend our framework with more types of choreography protocols, in particular protocols that will ease cancellation of services and allow for compensation mechanisms. We are also going to develop a more sophisticated model of brokerage and price forming.

References

1. W. M. P. van der Aalst. The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W. M. P. van der Aalst, M. Beisiegel, K. M. van Hee, D. König, and C. Stahl. A SOA-Based Architecture Framework. Computer Science Report 07/02, Technische Universiteit Eindhoven, The Netherlands, mar 2007.
3. A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, C. K. Liu, R. Khalaf, D. Knig, M. Marin, V. Mehta, S. Thatte, D. v. d. Rijn, P. Yendluri, and A. Yiu. Web Services Business Process Execution Language Version 2.0. Committee Draft, 25 January, 2007, Organization for the Advancement of Structured Information Standards (OASIS), Jan. 2007.
4. M. Beisiegel, H. Blohm, D. Booz, M. Edwards, O. Hurley, S. Ielceanu, A. Miller, A. Karmarkar, A. Malhotra, J. Marino, M. Nally, E. Newcomer, S. Patil, G. Pavlik, M. Raepple, M. Rowley, K. Tam, S. Vorthmann, P. Walker, and L. Waterman. Service Component Architecture – Assembly Model Specification. SCA Version 1.00, March 15 2007, IBM, SAP et al., mar 2007.
5. J. v. Brocke and M. A. Lindner. Service portfolio measurement: a framework for evaluating the financial consequences of out-tasking decisions. In M. Aiello, M. Aoyama, F. Curbera, and M. P. Papazoglou, editors, *ICSOC 2004*, pages 203–211. ACM, 2004.
6. J. Cardoso, A. P. Sheth, J. A. Miller, J. Arnold, and K. Kochut. Quality of service for workflows and web service processes. *J. Web Sem.*, 1(3):281–308, 2004.
7. R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Candidate Recommendation 27 March 2006, W3C, 2006.
8. O. Günther, G. Tamm, and F. Leymann. Pricing web services. In F. Leymann, W. Reisig, S. R. Thatte, and W. van der Aalst, editors, *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.
9. P. Massuthe, W. Reisig, and K. Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005.
10. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr. 1989.
11. F. D. Paoli, G. Lulli, and A. Maurino. Design of quality-based composite web services. In A. Dan and W. Lamersdorf, editors, *ICSOC 2006*, volume 4294 of *Lecture Notes in Computer Science*, pages 153–164. Springer, 2006.
12. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, Berlin, Germany, 1985.
13. H. M. W. Verbeek, T. Basten, and W. M. P. van der Aalst. Diagnozing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
14. H. M. W. Verbeek and W. M. P. van der Aalst. Woflan 2.0: A Petri-net-based workflow diagnosis tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 475–484. Springer, Berlin, Germany, 2000.
15. L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. Software Eng.*, 30(5):311–327, 2004.