QoS Aggregation for Web Service Composition using Workflow Patterns

Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muehl
Technische Universität Berlin
FG Intelligent Distributed Systems
Einsteinufer 17, Sek. EN-6
10587 Berlin, Germany
{mcj,gr,gmuehl}@ivs.tu-berlin.de

Abstract

Contributions in the field of Web services have identified that (a) finding matches between semantic descriptions of advertised and requested services and (b) non-functional characteristics – the Quality of Service (QoS) – are the most crucial criteria for composition of Web services. In this work a mechanism is introduced that determines the QoS of a Web service composition by aggregating the QoS dimensions of the individual services. This allows to verify whether a set of services selected for composition satisfies the QoS requirements for the whole composition. The aggregation performed builds upon abstract composition patterns, which represent basic structural elements of a composition, like sequence, loop, or parallel execution.

This work focusses on workflow management environments. This paper defines composition patterns that are derived from Van der Aalst's et al. comprehensive collection of Workflow Patterns. The resulting aggregation schema supports the same structural elements as found in workflows. Furthermore, the aggregation of serveral QoS dimensions is discussed.

1 Introduction

Using today's workflow management systems, services and components are manually integrated into a workflow. We can assume that the technical challenges of service integration in terms of compatibility and adaptation are already solved. The next emerging area in this domain is the automated integration of services enriched with approaches from the Semantic Web [15]. Thus, a higher level of interoperability must be ensured, so that software-systems can operate with the highest degree of autonomy possible. To realise a higher level of automation, it is not only sufficient to interpret the semantics of the integrated services or find matching for required interface signatures but also to take their Quality of Service (QoS) properties into account for aggregation. The interoperation of distributed software-systems is always affected by failures, dynamic changes, the availability of resources and others. These effects are non-functional aspects and caused by the nature of distributed software-systems [19]. A service that does not provide an acceptable QoS might be as useless as a service not providing the desired functional results.

For this work we focus on the composition of services with QoS properties, which is similar to integration process in workflow management systems. The goal of a composition is to have a collection of network resident services, whose functionality can be automatically discovered and integrated into applications. The common point of service composition and workflow management is that both presume the existence of a central peer, which is mediating the execution [12]. But there are also several differences, such as:

Variety For a composition description, more than one service may be available. They may vary in their functional and non-functional characteristics. Thus, a composition environment should be able to identify the best match by relevant criteria [22].

Dynamic Binding Assuming that most services for composition are based on Web Services technologies, they can be discovered and bound dynamically. That allows a composition environment to replace or even find services for the composition at runtime. But this flexibility needs to perform replacements or bindings safely with still ensuring the characteristics of the composition.

Only Web Services Contrary to existing workflow environments, the composition of Web Services does not

involve human-to-computer activities [4], but is oriented more to a messaging behaviour between the involved services.

Our approach is to apply the composition of services in the workflow domain for automated integration. In order to enable the selection of services and to leverage the advantages of the dynamic binding, the composition process must also take the QoS into account. We propose a mechanism to determine the QoS in a composition by using a generic workflow model as a basis for the algorithmic aggregation of local QoS dimensions.

The paper is organised as follows: in the first part, we discuss the patterns of workflow descriptions and workflow management systems. Then we will present how we derive a set of new patterns, which can be applied for compositions of Web services. Based on this, we explain, how the aggregation can be performed. Additionally, we define the aggregation for some QoS dimensions, like execution time or the cost of a service. The paper ends with a discussion about the related work and our conclusions.

2 Motivation

In a workflow management environment, a modeller will create an abstract workflow description, which is used for the identification and the integration of available services into the workflow. For the case that a workflow consists only of services, the result of the integration process can be seen as a composition of services. Adding the support for QoS in a composition could serve two purposes:

Variety Services must be selected referring to the global QoS requirements of the composition. Thus, local QoS values of available services must be aggregated to a global result.

Dynamic Binding When replacing, adding, or modifying a service in a composition, the global characteristic of the composition must be still ensured by the QoS requirements. Thus, an aggregation mechanism must determine the QoS of the composition after a local QoS has changed.

The Reference Model of the Workflow Management Coalition distinguishes functions of a workflow management system between *build time* - and *run time functions* [11]. According to this model, addressing the selection of services is a build-time functionality and supporting the dynamic binding is a run-time functionality.

3 Approach

Since the composition of Web Services is a relatively new field, different languages for the description of compositions are currently evolving. But rather than choosing a particular language we have defined an abstract model, which can be applied to different composition descriptions. This approach has following advantages:

- It is likely that the current specifications of the different languages may become revised. In this case an abstract model could be easier adapted to the changes.
- The currently proposed languages or the organisations behind them are currently in competition. The result could be that a certain language might become an industry leader or that some might disappear.

Our approach identifies abstract composition patterns, which represent basic structural elements of a composition, like a sequence, a loop, or a parallel execution. The patterns are used to model the structure of a composition. Since the motivation for this work is to use the composition of Web Services in the workflow domain, the composition patterns are based on structural elements as used in workflow descriptions.

Our proposal is to define the aggregation of QoS for each pattern and each type of QoS dimension. Using an hierarchical approach, the aggregated QoS dimensions for each composition pattern can be aggregated in order to provide QoS dimensions for the whole composition. For each combination of composition patterns and QoS dimensions one aggregation definition must exist. We have found some classification approaches for QoS dimensions as presented in QML [9], which is a language for the description of QoS using XML. In such languages QoS dimensions can be qualified by characteristics like direction, definition, or value type. But there are several reasons why such a classification is not beneficial for the aggregation:

- A general differentiation between increasing or decreasing directions for QoS dimensions exists. However, the definition of aggregation for one dimension is very different for either relative or absolute numerical values.
- QoS dimensions might have a direction but are discrete, so another aggregation mechanism might be necessary than with continuos dimensions.
- QoS dimensions might refer to a statistical definition - like variance, or the mean - so therefore in some parallel composition arrangements, a quite specific aggregation definition for each is needed.
- Some QoS dimensions are not numerical values, which can be mathematically aggregated. Another - i.e. rule-based - aggregation definition might be needed.

- The definition of QoS dimensions may vary depending on the environment. Thus, a classification of QoS dimensions for generalisation cannot be applied.

4 Aggregation

Since our basic use case is the composition of Web Services based on flow descriptions, we build on the existing work about workflow patterns of Aalst et al [21]. In the following we explain why we have chosen the approach based on workflow patterns and how we have derived our composition model from this work.

4.1 Workflow Patterns

The proposed workflow patterns were created to define a set of requirements for workflow management systems. They mainly describe their capability of executing different workflow structures and their behaviour. The advantage of these patterns lies in the ability to compare workflow management systems by their functional rather by their non-functional aspects (like platform requirements). Choosing the workflow patterns for our composition model has the following reasons:

- The workflow patterns were designed to provide an uniform approach for the comparison of workflow management systems and their flow definition languages. Thus, the patterns can be regarded as a comprehensive description about the characteristics of a workflow.
- Since the workflow patterns were published first in the year 2000, the work about the workflow patterns can be regarded as mature [2]. According to own statements in [21] some vendors of workflow management systems (e.g. COSA, FLOWer, or Staffware) refer already to the work about the workflow patterns.
- The existing flow languages for the composition of Web Services are based on a similar approach like modelling languages for workflows. Thus, a comparison of these languages based on the workflow patterns exists already [20]. This work also points out the strong relation between workflow management and service compostion.

In Table 1 we summarised the described workflow patterns. We have also identified the relevant patterns for the composition scenario. In our work only patters are relevant that address the structure of a workflow in the process modelling phase. For example, the pattern *Cancel Case* describes an execution pattern of a workflow management system and is therefore not relevant for the composition process. The identification for some patters is trivial (e.g. for the *sequence*). However other patterns require some discussion about the relevance for composition:

Multi Merge: This pattern describes a specific join operation of parallel executions in a workflow arriving at the joining point. At this point for each workflow execution arriving in parallel one following workflow is being executed. Since this feature can be implemented with just keeping different executions in parallel without synchronisation, these patterns can be represented by using AND split and an AND join operations for the aggregation.

Implicit Termination: The implicit termination pattern means that an engine is capable to terminate the flow, if no processable data is available anymore. Since the fact, whether an end is explicitly stated or ends implicitly, does not have an impact on the aggregation of QoS properties, this pattern is not taken into account.

Deferred Choice: The difference between a deferred choice in workflow context and the XOR split is that in the deferred choice the split is performed based on external input while the standard XOR split relies on information being part of the workflow. In the description about the workflow patterns, the deferred choice is referenced as a ”state based pattern”, thus not regarded for having an impact on the modelling phase.

Patterns Involving Multiple Instances: This set of patterns generally targets the run time abilities of a workflow management system. Therefore in the modelling phase each workflow or each composition can be seen as one unit for property aggregation. If at design time the number of instances is already known, this results in a parallel split (for patterns 12 and 13) with an open end. If the number of instances is determined during runtime (referring to patterns 14 and 15) the handling of QoS properties must be processed by the run-time environment of the composition and therefore is not taken into account in the modelling process.

Milestone: This pattern has got an impact on the execution on the workflow and is also taken into account for modelling the workflows. However, in the case of service composition we do not regards this pattern as being substantial. Furthermore this pattern is also not supported by any of the flow languages analysed in [20].

Cancellation Patterns: The cancellation patterns describe the ability of workflow management systems to cancel the execution of a workflow. Since this work targets the modelling phase of a workflow/composition, these patterns are regarded as not important for the composition model.

No.	Workflow Pattern	Synonym(s)	Rel.	Abstraction
Basic Control Flow Patterns				
1	Sequence	Sequential routing	Y	Sequence
2	Parallel Split	AND-split	Y	AND Split
3	Synchronisation	AND-join	Y	AND Join
4	Exclusive Choice	XOR-split	Y	XOR Split
5	Simple Merge	XOR-join	Y	XOR Join
Advanced Branching and Synchronisation Patterns				
6	Multi-choice	OR-split	Y	OR Split
7	Synchronising Merge	Synchronising join	Y	OR Join
8	Multi-merge		Y	AND Split with AND Join
9	Discriminator	m-out-of-n, partial join	Y	m-out-of-n
Structural Patterns				
10	Arbitrary Cycles	Loop, iteration, cycle	Y	Loop
11	Implicit Termination	Nothing to do anymore	N	
Patterns involving Multiple Instances				
12	M.I. Without Synchronisation		Y	AND Split with AND Join
13	M.I. With a Priori Design Time Knowledge		Y	represented by patterns 2-9
14	M.I. With a Priori Runtime Knowledge		N	
15	M.I. Without a Priori Runtime Knowledge		N	
State-based Patterns				
16	Deferred Choice	External choice, deferred XOR-split	Y	XOR Split
17	Interleaved Parallel Routing	Unordered sequence	Y	Sequence
18	Milestone	Test arc, state condition, withdraw message	N	
Cancellation Patterns				
19	Cancel Activity	Withdraw activity	N	
20	Cancel Case	Withdraw case	N	

Table 1. Workflow Patterns and their Relevance for Composition Patterns

Based on this analysis we have identified three relevant sequential composition patterns: the trivial sequence, an unordered sequence, and a loop. For the parallel case we have identified different parallel split and join operations, which can be summarised mainly in parallel AND, OR, and XOR split and join operations. We will discuss the concrete compositions patterns in the next section.

4.2 Composition Patterns

Based on the analysis of the patterns in the previous section, we can derive a composition model, which is the basis for the aggregation of QoS properties. To ensure that the aggregation results in relevant statements about the composition, our composition model is based on the following assumptions:

Independence: It is assumed that the services for a composition are not dependent on each other in their execution. This assumption presumes that the result or the execution of one service does not change the QoS definitions of other services.

Trust: For the aggregation of properties we assume that the given properties of a service are correct. We think that trusting the correctness of properties is a conceptionally separate issue from the algorithmic aggregation.

Uniformity: For the aggregation we assume the uniformity of the properties. This means that values refer to the same definition. For example we assume that a property describing the execution time is generally given in milliseconds, seconds, or minutes and it is assumed that all given values are generated by the same definition.

Equipartition: In the parallel case we assume for the aggregation of services an equal probability for each service in join and split cases. For example if in the case of an XOR split a service needs to be chosen among a number of services we assume all services have equal probabilities. Taking probabilities into account can be seen as an extension to the composition model.

In figure 1 the composition patterns are graphically shown. In the composition model two sequential patterns are defined:

- A simple sequence of service executions. The workflow pattern "arbitrary sequence" is also covered by this composition pattern, because it is assumed, that the successful execution of a service is independent form the execution of other services. (CP_1)
- A loop where the execution of a service or a composition of services is repeated for a certain amount of time. (CP_2)

For the parallel patterns we cannot only re-use the relevant workflow patterns, because for the algorithmic aggregation of values not only the split condition but also the join condition is relevant. For example, shall the mean execution time be subject for aggregation. If a flow splits into a number of parallel flows, two join structures could be possible: (a) joining with synchronisation of all parallel flows or (b) joining with synchronisation of one flow (workflow pattern "discriminator"). In case (a) the mean execution time would be the mean execution time of the greatest value (the slowest), and in case (b) the mean execution time is the mean of the given values (because of the assumption, that the discriminator will choose among the arriving services with an equal probability). The discriminator pattern denotes that after the synchronising flow has arrived the other flows are ignored.

There is also another reason for combining different split and join patterns: In the next section we will describe an aggregation scheme, which is basically realised with a backtracking algorithm, which requires a hierarchical structure of the composition. Therefore we need to divide the composition into independent atomic structures. It is thus obvious that various combinations between split and join operations are possible. However, only some of them match. For example an XOR split cannot occur in combination with an AND join. We have identified the following relevant composition patterns for the parallel case:

- XOR split followed by a XOR join. (CP_3)
- AND split followed by an AND join (CP_4)
- AND split followed by a m-out-of-n join (discriminator). (CP_5)
- OR split followed by OR join. (CP_6)
- OR split followed by a m-out-of-n join (discriminator). (CP_7)

The result is now a composition model based on patterns, which are derived from the workflow patterns identified in languages for Web Service composition. From now on a description in BPEL4WS for example can be transferred into this model, which is the basis for the aggregation. In the following section we explain how this aggregation is performed.

4.3 Aggregation Schema

The composition patterns we have proposed anticipate that the described flow can be represented using *directed graphs* which specify the order in which activities are executed. For aggregation, we propose to stepwise collapse

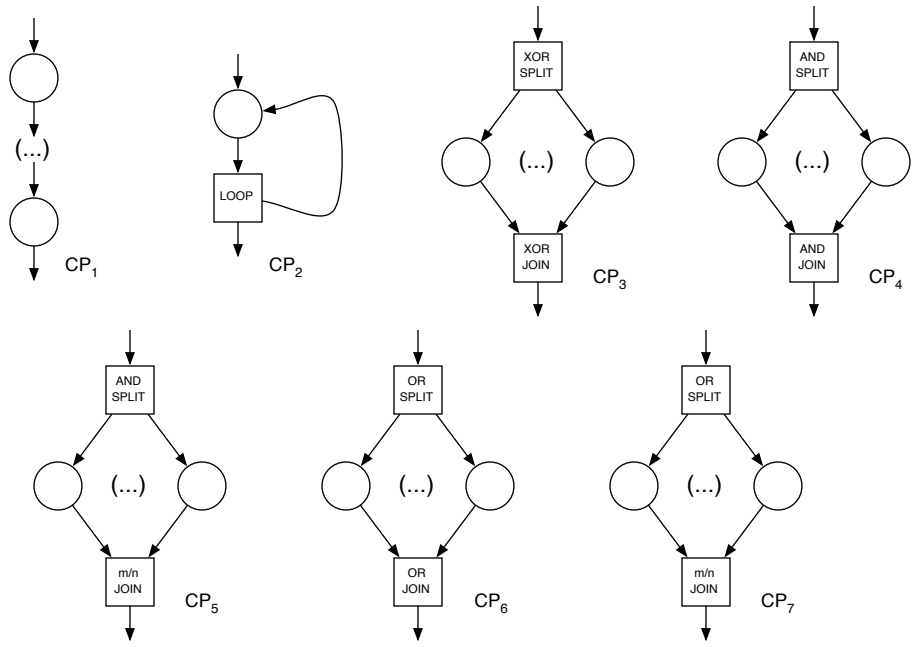

Figure 1. Different Patterns of the Composition Model

the graph into a single node by alternately aggregating simple sequences and parallel service executions (see Figure 2). This approach enables us to view aggregation in the following from a *micro-perspective*, i.e., not the whole graph is relevant at once for the aggregation process, but rather only local *composition patterns* which are accordingly differentiated into sequential and parallel service executions.

Applying this procedure, the aggregation can be performed on the basis of each composition pattern. The result is a composition model that has got the following key dimensions as explained in [12]:

- It assumes that each peer service has got a *bound queue* with the length of 1. This conforms to the characteristic of a flow where a peer is blocking the flow, when processing the desired message/data.

- The topology is *mediator based* - contrary to a brokered approach. Furthermore, it is obvious that the flow has a tree-oriented hierarchy, otherwise we would not be able to collapse a local composition pattern.

- The composition model can be applied to compositions that consider both *closed and open environments*. An open system allows the integration of services from an external environment, and thus depends on the tools used for composition and organisational constrains of using external services.

Based on this model the aggregation of numerical QoS dimensions can now easily be performed. We have summarised the aggregation schemes in tables 2, 3 and 4. The following notation is used: n is the number of services, k is the assumed number of repetitions in the loop, x_n is a given value for each service, and $\mathbb{N}$ is the set containing all x_n. The variable x_a represents the aggregated value.

In this model the loop case is not solved sufficiently, because it is likely that the number of loops cannot be determined during the design time (because they depend on conditions that will occur during runtime). For example, in BPEL4WS the condition of a loop statement is defined with a bool-expression (the while-construct), the proposed aggregation covers only the case of knowledge about the number of repetitions at design time. For other scenarios a different aggregation pattern must be implemented. For the loop case the model would fit well in monitoring QoS properties for analysis, but cannot serve as a decision support in the modelling process.

To address the OR splits, an aggregation model must know which paths are taken into account for this split. Keeping the assumption that equal probabilities among all choices hold, let set $\mathbb{T}$ contain all sets of the power-set of $\mathbb{N}$ that are relevant for the OR split. To determine the upper and lower bound for QoS values all combinations of possible splits must be taken into account. This results in the power-set $\mathcal{P}(\mathbb{T})$. To simplify the notation in the tables we define that for an aggregation operation applied to the ele-

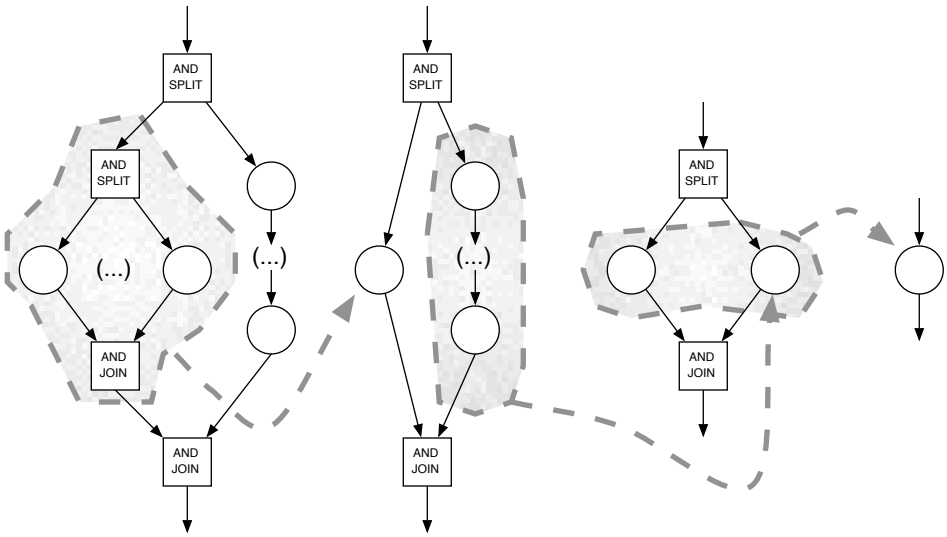

Figure 2. A Stepwise Graph Transformation.

ments of a power set:

$$f(\mathcal{P}(\mathbb{T})) := f\Big(\big\{x : x = f(\mathbb{S}), \forall \mathbb{S} \in \mathcal{P}(\mathbb{T})\big\}\Big)$$

For example, if the maximum of all values of $\mathbb{T}$ is desired, the maximum operation is first applied to each element of $\mathcal{P}(\mathbb{T})$, which is resulting in a new set. For this new set the maximum operation is applied as well.

Execution Time Applying our scheme to the aggregation of execution times, a definition for lower and upper bounds is shown in table 2. In a sequence, the time is determined by the sum of the values of each involved service. The definitions for lower an upper bounds are in a sequential case the same. Because the time is a decreasing measure in the sense of QoS, the largest value in a parallel arrangement of services denotes the worst case. This case demonstrates that with the same split operation (AND) it depends on the type of synchronising join operation, how the minimal execution time is determined. To determine the minimal execution time in the AND-AND case, the worst/largest value of all involved services denotes the overall minimal value.

Cost The cost of a service is a measure for the resources consumed by a service execution. Therefore, the cost can be seen as a decreasing dimension, where a lower value denotes a better service. Table 3 shows the aggregation definitions for the upper and lower bounds of the cost. Contrary to the execution time, all services must be taken into account that were started, regardless whether they are relevant for the synchronising join or not.

Encryption Level In this discussion it is assumed that the encryption level is equivalent with the length of a key, that is used for signing or encrytion. Generally it denotes the level of protecting the data and is thus an increasing dimension. Usually, key lengths are given in bits. For the aggregation of the encryption level in sequential patterns the weakest key is only significant.

In the parallel case the encryption level is a non-functional characteristic, which must be fulfilled by all significant parallel nodes in the same manner. But we can assume that if the node does not fulfill a demanded level of encryption, the execution would be worthless. Thus, this dimension has got an discrete character and aggregation based on for example the average of involved services cannot be applied. The aggregation for the encryption is listed in table 4.

Throughput The throughput of a service denotes the amount of processable data per time unit. Usually, the throughput is given in $bytes/sec$ and is interpreted as an increasing dimension. In a sequence, the node with the lowest value assigned determines the throughput for the aggregated property. In a parallel arrangement, the throughput of each starting node must be considered, because exceeding the defined throughput could result in an unsuccessful execution. The aggregation of throughput is shown in table 4

Uptime Probability The uptime probability denotes the probability that the execution of the node performs

successfully and is regarded as a decreasing dimension. For this dimension only the lower bound is relevant, because a maximal uptime probability does not make sense. The aggregation of the uptime probability is also shown in table 4.

Especially for this dimension, the assumption that each serve executes independently from others is relevant. Otherwise the model must be extended with a specialised case, that the uptime probability of interrelated services must be combined.

5 Related Work

There are several contributions addressing the composition of Web Services. A growing interest can be noticed in this field, because the composition of Web Services can enhance the execution of workflows. There are proposals for languages to describe compositions like BPEL4WS [6], WSCI [3], XLANG [18], or WSFL [13]. The composition of Web Services is also an issue for contributions about describing services with semantic information. As a complementary approach the semantic description of services can also be used as a criteria for the selection or dynamic binding of services as we have introduced using QoS dimensions. The W3C proposes DAML-S (renamed as OWL-S) [7] as an ontology for service description.

The contributions addressing the composition structures are not relying on a stringent structural model. A reference model proposed by Yang et al. [22] doesn't cover as many structural cases as we are considering in our model. For example, their model lacks the cases for a parallel execution scheme, where either m-out-of-n services are chosen for execution (so called OR-split) or m-out-of-n services are required for synchronisation (a so called discriminator). In his PhD thesis [5], Jorge Cardoso discusses the aggregation of QoS properties in a workflow for some specific QoS aspects like cost or response time. Contrary to our work, the composition model does not identify general structures, but analyses some specific application scenarios. So the proposed structures are limited in combination to their application case, and the model has also the flaws we discussed for the model of Yang et al.

An approach from another perspective can be seen with the introduction of *XL*, which is an XML-based programming language for "the specification of Web Services" [8]. In fact this approach allows to define a composition of Web Services using a specific programming language. The advantage of this approach is that a common type system and data model is used for both, the description of Web Services and the programming language. One design cornerstone in their proposal is the separation of declarative elements of Web Services from the operational expressions. In summary this work covers a lot of the aspects for which we have used the workflow patterns instead. However, for our approach building not onto XL has two main reasons: (a) we expect a bigger momentum with BPEL4WS or other flow languages to define Web Service compositions than with XL. And (b) we believe that we need to orientate our work to the likely scenarios of the industry application, which is the application in workflow management systems. Contrary to the flow languages we think that XL is not focussed on applications in this field.

Further examples addressing the composition of Web Services with custom composition patterns and custom aggregation models can be found in the work of Daniel A. Menasce [14], which also addresses the aggregation of QOS dimensions, or in [10]. But in summary all the proposed material does not address the composition structure in the way, that reflects the occurring patterns in workflows. Therefore in our work the relevant contribution is the definition of an abstract, workflow-oriented composition model to aggregate of QoS properties.

Puschner and Schedl [17] also used a graph-based approached to aggregate the execution times of atomic executions units in real-time systems. A work from the field of real-time based systems has already used a graph-based approached to aggregate the execution times of atomic executions units in real-time systems. They use the same principles for the aggregation of execution times of software units as we do. Besides that their work is remarkable for the design of real-time systems, our composition model distinguishes more cases in order to address the workflow patterns.

6 Conclusions

We have presented a composition model that can serve as a basis for aggregation of service properties for QoS dimensions. The model assumes that for the composition of Web Services that use common flow languages standard workflow patterns apply. Therefore workflow patterns of Aalst et. al are used to derive the introduced composition patterns which enable the aggregation of properties. We have modelled a flow description of a Web service composition as a graph, and we have transformed this graph into a graph of composition patterns. With this graph of composition patterns we have used a recursive approach to aggregate properties on the level of identified composition patterns.

The composition model based on common workflow structures as presented in the workflow patterns delivers a stringent and clean model for the aggregation, which can be implemented. The given definitions for aggregating values according to the composition can be used to extend a tool for modelling Web service compositions. We see the next steps in the integration of trust aspects when properties are retrieved and the integration in a tool for modelling Web ser-

#	Comp. Pattern	Max. Execution Time	Min. Execution Time
Sequential Composition Patterns			
1	Sequence	$x_a = \sum_{i=1}^{n} x_i$	$x_a = \sum_{i=1}^{n} x_i$
2	Loop	$x_a = kx$	$x_a = kx$
Parallel Composition Patterns			
3	XOR-XOR	$x_a = max\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$
4	AND-AND	$x_a = max\{x_1, \ldots, x_n\}$	$x_a = max\{x_1, \ldots, x_n\}$
5	AND-DISC	$x_a = max\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$
6	OR-OR	$x_a = max\big(\mathcal{P}(\mathbb{T})\big)$	$x_a = min\big(\mathcal{P}(\mathbb{T})\big)$
7	OR-DISC	$x_a = max\big(\mathcal{P}(\mathbb{T})\big)$	$x_a = min\big(\mathcal{P}(\mathbb{T})\big)$

Table 2. Aggregation of Numerical QoS Dimensions: Upper and Lower Bounds of Execution Time

#	Comp. Pattern	Max Cost	Min Cost
Sequential Composition Patterns			
1	Sequence	$x_a = \sum_{i=1}^{n} x_i$	$x_a = \sum_{i=1}^{n} x_i$
2	Loop	$x_a = kx$	$x_a = kx$
Parallel Composition Patterns			
3	XOR-XOR	$x_a = max\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$
4	AND-AND	$x_a = \sum_{i=1}^{n} x_i$	$x_a = \sum_{i=1}^{n} x_i$
5	AND-DISC	$x_a = \sum_{i=1}^{n} x_i$	$x_a = \sum_{i=1}^{n} x_i$
6	OR-OR	$x_a = max\Big\{z : z = \sum_{y \in \mathbb{S}} y, \forall \mathbb{S} \in \mathcal{P}(\mathbb{T})\Big\}$	$x_a = min\Big\{z : z = \sum_{y \in \mathbb{S}} y, \forall \mathbb{S} \in \mathcal{P}(\mathbb{T})\Big\}$
7	OR-DISC	"	"

Table 3. Aggregation of Numerical QoS Dimensions: Upper and Lower Bounds of Cost

vice compositions. Also, the composition model could be discussised/extended for the case that one of the mentioned assumptions does not hold.

For future extensions of our composition model we will evaluate the contributions of modelling workflows or compositions of Web Services with Petri Nets as found in [1] and [16]) or with other formalisms like using linear temporal logic as found in [12]. This reference also introduces a composition framework, which we will address in our model in section 4.3.

Acknowledgments

The authors wish to thank Kurt Geihs, Andreas Tanner, Gerhard Koehler and Torben Weis for many fruitful discussions.

References

[1] W.M.P. van der Aalst, K.M. van Hee, and G.J. Houben. Modelling workflow management systems with high-level petri nets. In G. De Michelis, C. Ellis, and G. Memmi, editors, *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50, 1994.

[2] W.M.P. van der Aalst, A.H.M. ter Hofstede, A. Kiepuszewski, and A.P. Barros. Advanced workflow patterns. In *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 18–29. Springer-Verlag, Berlin, 2000.

[3] Assaf Arkin et al. Web service choreography interface (wsci) 1.0. Technical report, W3C,

#	Description	Encrytion	Throughput	Uptime Probability
Sequential Composition Patterns				
1	Sequence	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = \prod_{i=1}^{n} x_i$
2	Loop	$x_a = x$	$x_a = x$	$x_a = x^k$
Parallel Composition Patterns				
3	XOR-XOR	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$
4	AND-AND	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = \prod_{i=1}^{n} x_i$
5	AND-DISC	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = \prod_{i=1}^{n} x_i$
6	OR-OR	$x_a = min\big(\mathcal{P}(\mathbb{T})\big)$	$x_a = min\big(\mathcal{P}(\mathbb{T})\big)$	$x_a = max\Big\{z : z = 1 - \prod_{y \in \mathbb{S}} (1-y), \forall \mathbb{S} \in \mathcal{P}(\mathbb{T})\Big\}$
7	OR-DISC	$x_a = min\big(\mathcal{P}(\mathbb{T})\big)$	$x_a = min\big(\mathcal{P}(\mathbb{T})\big)$	”

Table 4. Aggregation of Numerical QoS Dimensions: Encryption, Throughput and Uptime Probability

http://www.w3.org/TR/wsci, 2002.

[4] Gregory Alan Bolcer and Gail Kaiser. Swap: Leveraging the web to manage workflow. In *IEEE Internet Computing*, pages 85–88. IEEE, January-February 1999.

[5] Jorge Cardoso. *Quality of Service and Semantic Composition of Workflows*. PhD thesis, Department of Computer Sceince, Universoty of Georgia, Athens, GA (USA), 2002.

[6] Satish Tatte (Editor). Business Process Execution Language for Web Services Version 1.1. Technical report, BEA Systems, IBM Corp., Microsoft Corp., http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2003.

[7] Anupriya Ankolenkar et al. Daml-s: A semantic markup language for web services. In *Proceedings of 1st Semantic Web Working Symposium (SWWS' 01)*, pages 441–430, Stanford, USA, August 2001. Stanford University.

[8] Daniela Florescu, Andreas Gruenhagen, and Donald Kossmann. XL: an XML Programming Language for Web Service Specification and Composition. In *Proceedings of the eleventh international conference on World Wide Web*, pages 65–76. ACM Press, May 2002.

[9] Svend Frolund and Jari Koistinen. Quality of service specification in distributed object systems design. *Distributed Systems Engineering Journal*, 5(4), December 1998.

[10] Dinesh Ganesarajah and Emil Lupu. Workflow-based composition of web-services: a business model or a programming paradigm? In *Proceedings of Sixth International Enterprise Distributed Object Computing Conference, 2002. EDOC '02.*, pages 273–284. IEEE, September 2002.

[11] David Hollingsworth. The Workflow Reference Model. Technical Report TC00-1003, Workflow Management Coalition, 1995.

[12] Richard Hull, Michael Benedikt, Vassilis Christophides, and Jianwan Su. E-services: A look behind the curtain. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2003)*, San Diego, USA, June 2003. ACM Press.

[13] Frank Leymann. Web services flow languageweb services flow languageweb services flow languageweb services flow language. Technical report, IBM Software Group, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2001.

[14] Daniel A. Menasce. Qos issues in web services. In *IEEE Internet Computing*, pages 72–75. IEEE, November-December 2002.

[15] Eric Miller. Semantic web activity statement. Technical report, W3C, http://www.w3.org/2001/sw/Activity, 2003.

[16] Srini Narayanan and Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the eleventh international*

conference on World Wide Web, pages 77–88, Honolulu, USA, May 2002. ACM Press.

[17] Peter Puschner and Anton Schedl. Computing maximum task execution times - a graph-based approach. *Journal of Real-Time Systems*, 13(1):67–91, July 1997.

[18] Satish Thatte. Xlang - web services for business process design. Technical report, Microsoft Corporation, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, 2001.

[19] Andreas Ulrich, Torben Weis, Kurt Geihs, and Christian Becker. DotQoS - QoS extension for .NET Remoting. In K. Jeffay, I. Stoica, and K. Wehrle, editors, *International Workshop on Quality of Service (IWQoS)*, pages 363 – 380, Monterey, CA, June 2003. Springer-Verlag Heidelberg.

[20] W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. *Jan/Feb 2003 issue of IEEE Intelligent Systems*, pages 72–76, January 2003.

[21] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases 14(3)*, pages 5–51, 2003.

[22] Jian Yang, Mike P-Papazoglou, and Willem-Jan van den Heuvel. Tackling the challenges of service composition in e-marketplaces. In *Proceedings of the 12th International Workshop on Research Issues in Data Engineering (RIDE '02)*, pages 125–133. IEEE, February 2002.