# Model-Driven Methodology for Building QoS-Optimised Web Service Compositions

Roy Grønmo[1] and Michael C. Jaeger[2]

[1] SINTEF ICT, P.O.Box 124 Blindern N-0314 Oslo, Norway
Roy.gronmo@sintef.no
[2] TU Berlin, FG FLP SEK FR6-10, Franklinstrasse 28/29 D-10587 Berlin, Germany
mcj@cs.tu-berlin.de

**Abstract.** As the number of available Web services increases there is a growing demand to realise complex business processes by combining and reusing available basic Web services. For each of the needed basic Web services there may be many candidate services available from different vendors and with different Quality of Service (QoS) values. This paper proposes a model-driven methodology for building new Web service compositions that are QoS-optimised. We investigate if UML is suitable for modelling the QoS aspects and we explain how transformations can be used to automate most parts of the methodology. Some of these transformations are already implemented in previous work.
As part of the methodology we present a control flow pattern approach that optimises the QoS values of the composition given user-defined requirements and preferences. The pattern approach is compared towards the local and global approaches identified by other authors. Experiments are carried out within our simulation tool that simulates the selection of services and aggregation of QoS in a given composition. The test results are used to identify recommendations of when to use the different approaches based on measurements of computation time of the optimisation component and achieved QoS values for the new composition. The methodology is explained by relating it to a gas dispersion emergency case.

## 1 Introduction

A growing number of Web services are implemented and made available internally in an enterprise or externally for other users to consume. Such Web services can be reused and composed together in order to realise larger and more complex business processes. We define Web services to be services available by using Internet protocols such as HTTP and XML-based data formats for their description and invocation. A Web service composition is a description of how basic Web services can interoperate in order to accomplish a larger task. There is not yet any de-facto standard for defining Web service compositions although there are several evolving proposals (BPEL4WS [3] etc). All of these composition languages, as well as the other adopted Web service specifications (SOAP, WSDL) use low-level XML code as the representation language. We propose to use a model-driven methodology that provides a relation between the XML specifications and graphical, higher-level models using the Unified Modelling Language (UML). This improves the development and maintenance of new Web services.

In a composition there are many sub-tasks that need to be solved by calls to existing services. There may be many alternative services that can accomplish the same task,

but with different QoS offerings such as price and data quality. This means that a selection among these services must be taken. For the automatic selection of services and computation of overall QoS values, we introduce a control-flow pattern approach. Two main hypotheses are elaborated in the paper:

- The forthcoming UML Profile for modelling QoS from the Object Management Group (OMG) [9] can be used to capture the necessary QoS aspects in UML for Web service composition.
- The pattern approach to QoS-optimisation achieves improved overall QoS values of the composition and/or improved computation time compared to previously published approaches.

Both hypotheses are evaluated against a gas dispersion emergency case. The modelling constructs of OMG's UML profile is applied to this case to see if they are suitable and cover all the aspects relevant in this case. The pattern approach and the alternative approaches are simulated by a tool that compares the achieved overall QoS values of the whole composition and the computation time of the algorithms. The tests use the fixed composition of the gas dispersion case and randomly generated candidate services for each task in the composition with randomly assigned QoS values.

This paper is organised as follows. Section 2 provides an overview of the methodology. Sections 3 to 6 go through the four methodology steps in detail as applied to the gas dispersion case. Section 7 discusses assumptions and potential shortcomings in our current methodology. Section 8 describes a simulation tool that is used to compare our optimisation approach with two other approaches. Section 8 presents related work and section 9 summarises with conclusions and by identifying future work.

## 2 Methodology

This section describes a methodology for designing and implementing QoS-optimised Web service compositions. The methodology is model-driven using UML as the modelling language. We propose to define and implement fully automated transformations whenever applicable. The main focus in our methodology is to choose the most appropriate web services based on their QoS offerings. Therefore QoS offered values need to be retrieved somehow by the service requester. Since there is no standardised way to do this, we shortly explain three alternative ways such information may be retrieved:

- Extending WSDL with a reference to offered QoS (Figure 1). This enables each WSDL operation to be associated with a QoS offerings document which may be updated regularly by the service provider if the values change.

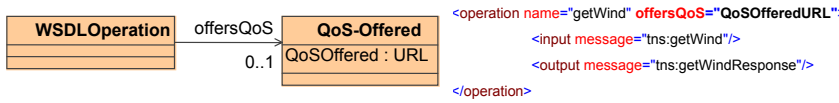


**Fig. 1.** Extending WSDL to provide QoS information

- The service requester negotiates a QoS contract with the service providers. Signed bilateral contracts define the QoS offerings that the service provider is obliged to deliver. The negotiation process may be automatic or manual.
- Asking QoS testing services. A QoS testing service will calculate QoS offered values by running test scenarios against the services, or it may gather information based on reports from other consumers of a service.

Especially with the first and last of these alternatives it will be a question of reliability of the retrieved QoS values. It is outside the scope of this paper to further investigate these alternatives. In the remainder of this paper we assume that the QoS values may be retrieved and be relied upon, and that the WSDL extension mechanism is used. Services without retrievable QoS values must be ignored as candidate services. Note that the QoS requirements will refer to a set of QoS characteristics, and QoS values for the same characteristics need to be specified in the offered QoS of the candidate service.

The methodology has four steps (Figure 2); In step 1 the user designs a composite Web service with a set of QoS requirements; In step 2 an optimisation service is used to select the best suited candidate services for the composition; In step 3 the model is finalised with details of chosen services and computed QoS values for the composition; In step 4 the model is exported into specifications that are used to implement, deploy and publish the composite Web service with its offered QoS values.

We will explain the methodology by applying it to a concrete example. The gas dispersion emergency case of the ACE-GIS project [13] has been extended with QoS aspects. The case is about how to handle an emergency situation caused by an explosion in a chemical factory, with subsequent leakage of poisonous gas. In order to make efficient evacuation plans, the crisis management needs a forecast of the gas plume dispersion. Hence a Web service needs to be designed which calculates the gas plume and displays it on a map. We introduce some QoS values for the different services in the case, but these are only provided as examples and are not grounded by any actual measurements.

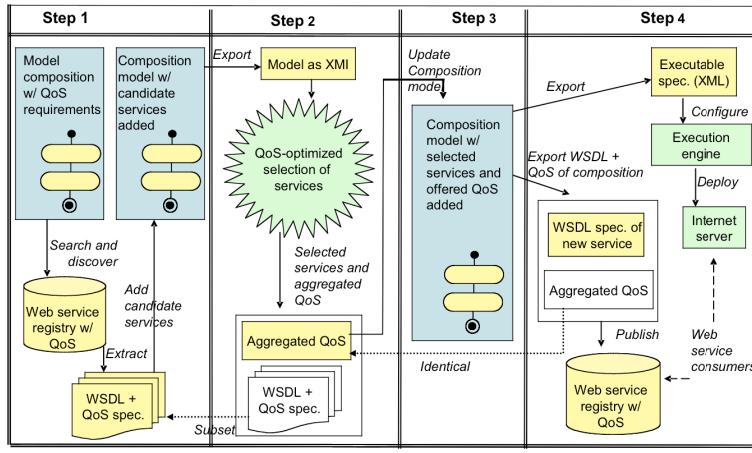


**Fig. 2.** Methodology

## 3 Step 1: Modelling the Composition with QoS Requirements

This section shows how UML can be used to model QoS requirements within a Web service composition. The QoS modelling builds upon OMG's forthcoming UML profile for QoS modelling [9].

**Modelling the Interface and Behaviour.** The first step is the interface modelling which identifies the new operations of the new composite Web service. For the gas dispersion example we wished to develop a Web service with one operation: CreateGasDispersionMap( plantID: string, emissionRate: float): base64Binary. By calling this operation with a plant identifier and an emission rate of the gas dispersion, the result will be an image map displaying the expected gas dispersion after a given time period on a background area map. For the identified operation we need to model the behaviour. This can be achieved by using UML activity models that capture the control flow and data flow aspects. Figure 3 shows a simplified activity model of the control flow for the CreateGasDispersion operation. First we get the location of the plant with gas leakage, and then the nearest airport is located since this is required by the Get Wind service[3]. In parallel a gas dispersion plume is calculated and a background map is retreived. Finally, a map with the gas dispersion plume overlaid on the background map is created.

**Modelling QoS Characteristics.** The QoS concepts need to be precisely defined and used by all the parties involved. The OMG profile uses QoS characteristics to define collections of QoS concepts with precise semantic meaning. Each QoS characteristic contains a set of QoS dimensions with a name and allowed value domain. A QoS dimension also has a direction which is defined as either increasing or decreasing, where increasing means that higher values are preferred. The QoS characteristics for the gas dispersion case are shown in Figure 4. A price is either given as monthly subscription or per call, and the currency is Euro. Execution time is measured in seconds and only with respect to worst case. A user rating specifies the user satisfaction on a scale from 1 to 10 where higher values are preferred. The encryption level ranges from 1 to 4 where higher values are preferred. These levels should be further detailed in order to be precisely defined.

**Modelling QoS Requirements.** When the QoS characteristics are defined, they can be used to model the QoS requirements. A QoS requirement identifies restrictions on a service with respect to the value domain of specific QoS characteristics. These requirements need to be fulfilled in order to achieve a successful binding to a service. The requirements may apply to a single service within the composition or to the service composition as a whole.

We propose to extend the OMG profile with QoS interests, which is the set of QoS characteristics that are desirable to be optimised. The QoS interests follows the same notational principles as the other parts of the OMG profile, such as the requirements modelling. A possible strategy is to implicitly include all of the QoS characteristics, referred by QoS requirements, as part of the interests set. In many cases this may not be desirable since optimization deals with trade-offs. Any additional characteristic that

[3] In the gas dispersion case, no Web service was found to provide a wind estimate for an arbitrary location. However, a Get Wind service taking an airport location as input, was found.

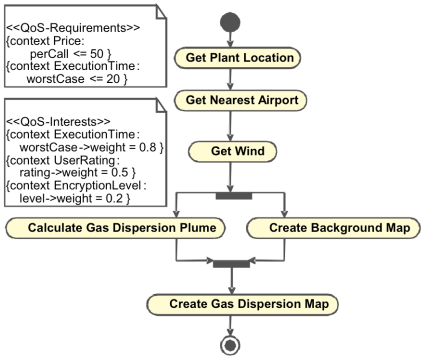


**Fig. 3.** Modelling QoS Requirements

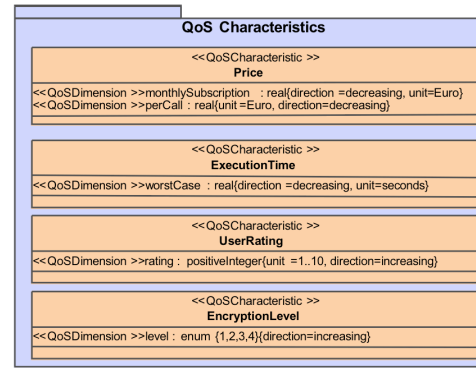


**Fig. 4.** Modelling QoS Characteristics

one tries to optimise may worsen the optimisation of the other characteristics. We instead recommend explicitly modelling all of the characteristics to be optimised within the interests set. The result may then be that characteristics specified only in the requirements set and not in the interests set will not be optimised any further beyond fulfilling the requirements upper and lower bounds. Furthermore we propose that the different characteristics in the interests set are weighted to indicate the relative importance of the interest.

Figure 3 shows the composition of the gas dispersion case with QoS requirements. The QoS requirements in the example (top left note) apply to the composition as a whole, since they are not attached to any the individual services (this is our notational convention). The requirements state that the required price per invocation must be at most 50 Euros and the execution time must never exceed 20 seconds. In addition we specify QoS interests (lower left note) indicating which QoS characteristics we want optimised. Each of these interests has assigned a weight between 0 and 1, where 1 indicates highest importance. The QoS characteristics used are already defined by a separate UML package as shown in Figure 4.

**Model with Candidate Services.** When searching for services we have ideally discovered candidate services for all the different tasks in our composition model. If not, two possible actions must be taken. Either the composition must be changed or the missing service must be implemented by ourselves. Each Web service is defined syntactically by the de-facto standard WSDL. Updating the composition model with the candidate services can be supported by reverse transformations from WSDL to UML [5]. We also need to attach these candidate services to its belonging task. The metamodel for attaching candidate services to the tasks are given in Figure 5. Each task can be realised by many candidate services. A candidate service is defined by a WSDL operation. Note that one WSDL file may provide several operations. The four attributes wsdlFile, wsdlOperation, wsdlPortType and wsdlService will together give a unique reference to the WSDL operation. The WSDL operation refers to a document describing its offered QoS.

The modelling of candidates in UML can be handled by letting each action have a tagged value, CandidateServices, which defines the URL of an XML file with a list of

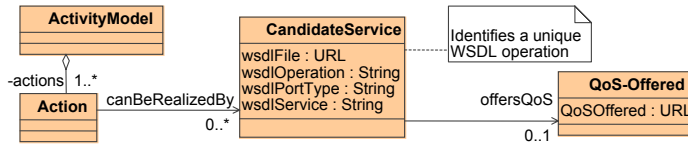


**Fig. 5.** Metamodel for Candidate Services

the candidate WSDL operations that can perform the task. Since there may be many candidate services for a task, this list structure is more technically feasible to capture in an XML file than in the UML model itself. The production of such an XML file can preferably be supported by a tool. The composition model with candidate services are sent to the QoS-optimisation algorithm discussed in the next section. The outcome of applying the algorithm will be one selected service for each task in the composition.

## 4 Step 2: QoS-Optimised Selection

In order to select candidates for tasks in a composition based on QoS requirements, a method is necessary to determine the QoS of a composition depending of the offered QoS of potential candidates. In a preceding paper we have introduced a set of seven structural elements – called *composition patterns* – to model a composition [7]. The composition patterns were derived from a set of workflow patterns, which have been introduced by van der Aalst et al. [12]. Workflow patterns form a set of functional and structural requirements for workflow management systems. We have chosen the workflow patterns as the foundation for the composition patterns because existing flow languages for the composition of Web services are based on a similar approach like modelling languages for workflows. Van der Aalst has also shown that the workflow patterns apply to existing composition languages [11]. The patterns are basic sequential and parallel arrangements of tasks and thus can be seen as the atomic elements of a composition:

**Sequence of service executions.** A sequence can be ordered or in arbitrary order. For the aggregation model, the order of the service executions is not relevant.

**Loop.** The execution of a service or a composition of services is repeated for a certain amount of times.

**XOR split followed by a XOR join.** From a parallel arrangement only one task is started. Thus, the synchronising operation only waits for the started task.

**AND split followed by an AND join.** From a parallel arrangement all tasks are started, and all tasks are needed to finish for synchronisation.

**AND split followed by a m-out-of-n join.** From a parallel arrangement all tasks are started, but less than all tasks are needed to finish for synchronisation.

**OR split followed by OR join.** In a parallel arrangement some of the available tasks are started and all of the started tasks are needed to finish for synchronisation.

**OR split followed by a m-out-of-n join.** In a parallel arrangement some of the available tasks are started but only a subset needed to finish for synchronisation.

The composition patterns anticipate that the described flow can be represented using directed graphs that specify the order in which activities are executed. For this example

this means that the XMI output, representing the UML model, is transformed into an internal graph representation that consists of the pattern elements. Then for aggregation, we propose to stepwise collapse the graph into a single node by alternately aggregating the sub-patterns starting with sub-patterns that do not contain any further sub-patterns. For the example shown in Figure 6, it means that two aggregation steps are performed: first the parallel arrangement and then the sequence containing the aggregated result from the parallel arrangement is aggregated. This results in a single QoS statement for each of the different QoS characteristics.

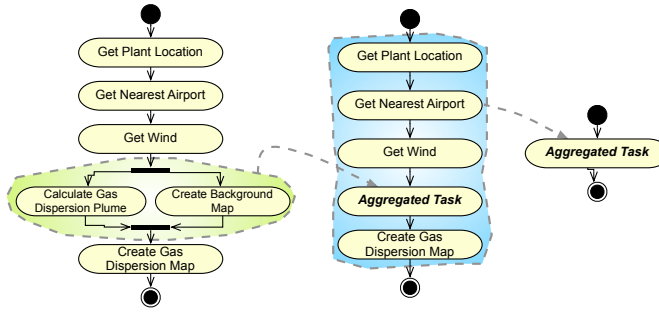


**Fig. 6.** Example of Collapsing the Graph for Aggregation

In summary, this approach enables us to view the aggregation in a pattern perspective, i.e., not the whole composition is relevant at once for the aggregation process, but rather only the local composition patterns, which are accordingly differentiated into sequential and parallel service executions. To cover the OR splits, an aggregation model must know which paths are taken into account for this split. To determine the aggregation of the QoS values, all combinations of possible splits must be taken into account.

**Worst case execution time.** For the sequential patterns the execution times are added, in the parallel cases the largest value is relevant for giving an estimation about the possible worst case.

**Price.** For the aggregation of the price in parallel arrangements, every started task is relevant. For parallel split cases, the maximum price possible is relevant.

**User rating.** The user rating can be aggregated by two approaches. Calculating the mean of the given values within an element performs the aggregation of the rating.

**Encryption.** The encryption case is very straightforward and considers that for a composition the chosen candidate that is offering the weakest encryption is relevant for the whole composition.

To apply the introduced aggregation rules and the aggregation scheme using the pattern-perspective approach some assumptions must remain valid, which we will discuss in section 7. With pattern-wise aggregation a method is available to calculate the QoS of the whole composition or its parts. Now, this method can be applied in a selection process that needs to evaluate the resulting QoS for different combinations of assigning candidates to tasks.

**Pattern-wise Selection.** In the selection process, for each task an available candidate is assigned. The simplest approach, called local-based, is to select a candidate by comparing the QoS among all candidates for a particular task and chose the candidate

that provides the best QoS. This selection algorithm shows a greedy behaviour. However, the QoS for a composition can be increased if an algorithm would perform the selection by considering the QoS of candidates for other tasks as well: consider our setup in the gas dispersion example. In this example, the parallel execution of the two tasks Calculate Gas Dispersion Plume and Create Background Map is given. Let the optimisation goal be to find the quickest execution while trying to keep the lowest price. Also, choosing a quicker service is usually more costly. For the case that even the quickest candidate for the left task executes longer than any of the candidates for the right task, the optimal assignment for the right task is the cheapest service, regardless how long its execution would take. Due to its nature, this kind of optimisation cannot be performed with a local-based approach.

To address this particular case, our selection algorithm evaluates the candidate services by evaluating the possible assignments within each pattern by performing the following steps:

1. In the graph of composition patterns, the algorithm walks by a recursive approach into the structure and identifies pattern elements that do not contain any sub-patterns. In the example this would be the element with the parallel arrangement.
2. For all tasks within this element, all combinations of candidate assignments are evaluated. For each combination the QoS is aggregated using the pattern-based aggregation and the combination that delivers the best overall QoS is chosen.

To evaluate the combinations among the elements the algorithm compares the QoS of either candidates or aggregated patterns by calculating a ratio between two sets of services. For this the *Simple Additive Weighting (SAW)* can be applied, which was introduced in the context of *Multiple Criteria Decision Making (MCDM)* [6]. By this approach for each QoS value a weight is applied and then a score is calculated for each candidate. The score is determined by firstly normalising the values among the candidates to compare. Let $s_{yx}$ be a single value with $x$ denoting the index of the candidate Web service and $y$ denoting a particular QoS category, then the normalised value $n_{yx}$ and the score for each candidate $x$ named $c_x$ are:

$$n_{yx} = \begin{cases} \frac{max_y(s_{yx}) - s_{yx}}{max_y(s_{yx}) - min_y(s_{yx})} & \text{for increasing categories} \\ \frac{s_{yx} - min_y(s_{yx})}{max_y(s_{yx}) - min_y(s_{yx})} & \text{for decreasing categories} \end{cases} \qquad c_x = \frac{1}{p} \sum_{y=1}^{p} w_y n_{yx}$$

The weight $w_y$ is applied to the categories by the user's preference and the value $p$ represents the number of used QoS categories. The sum of all weights must be equal to 1. The result of this procedure is a score by which the overall QoS of a candidate can be compared with the overall QoS of another.

The outcome of this phase is an assignment of exactly one service for each task in the composition and the resulting aggregated QoS statement that can be applied to the composition. Technically the result can be presented in one file for the QoS offered of the aggregation and one file that identifies the WSDL-operations assigned to identifiers of each task.

## 5 Step 3: Composition with Selected Services and Offered QoS

The composition model will be updated to show the selected services for each task including the individual offered QoS values of each service and the aggregated offered QoS for the composition as a whole. Figure 7 shows such an updated model for the gas dispersion case with imaginary offered QoS values. The production of this UML model is preferably supported by automatic reverse engineering in a transformation tool that takes the produced files from step 2 as input. The list of selected services will be assigned to its respective tasks and the details identifying the WSDL operation to execute is provided in the model as UML tagged values. To simplify the figure we have not shown these WSDL details, only the name of the provider of the service.

The offered QoS values are taken from the QoS offered documents assumed to accompany each WSDL operation. In UML we propose to specify the offered QoS values in a note attached to each task where the corresponding WSDL operation is chosen. The UML has also an unattached offered QoS note that by our convention indicates that it applies to the composition as a whole. The way of specifying the offered QoS value in the note follows the current draft OMG profile for QoS, while attaching them to the tasks and the activity model as a whole is a new way of applying the OMG profile. The current version of the OMG profile mentions only QoS offered for software components, classes or interfaces. The offered QoS values indicate the promised QoS for the individual services and the new Web service composition.

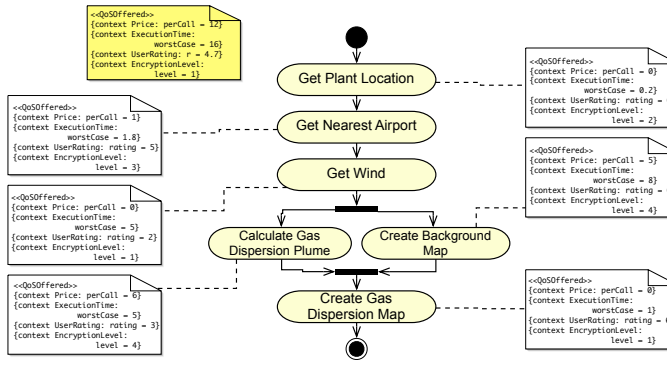


**Fig. 7.** Composition Model with Actual QoS Values

## 6 Step 4: Exporting the Model

The final Web service composition model with QoS offered is a basis for code generation to WSDL [5], execution specification (e.g. BPEL) [3] and a QoS XML document. Finally the Web service composition is deployed and published as WSDL with accompanying offered QoS values for the service composition as a whole. The WSDL of the

gas dispersion case consists of one operation CreateGasDispersionMap() with a reference to the QoS values. The QoS values in the XML file are equivalent to those in the model of Figure 7. The offered price per call is 12 Euro, the worst case execution time is 16 seconds, the user rating is 4.7, and the encryption level is 1.

## 7 Discussion

The Web service composition methodology is an iterative and continuously evolving process since the available services and their QoS values may change at any time. New services will come along, other services will have improved or worsened QoS, yet other services will no longer be available. In order to guarantee the offered QoS for the whole composition one needs to negotiate and agree on contracts between the party offering the composite service and each individual service party. When the composite service party is notified of a change to one of its sub-contractor's services, all the steps in the methodology of this paper should be reapplied as a new iteration.

To ensure that new, possibly QoS improving services are utilised, the methodology steps should be reapplied at a regular basis. The first step in the methodology is the only one that requires human intervention. This step can also be fully automated in the future, but this will require a lot of unsolved infrastructure support with respect to semantic properties and standardised use of semantic ontologies. Our methodology assumes that the WSDL operations have references to a document describing the offered QoS values. It is natural to add a language to capture this information to the family of Web service specifications such as WSDL, SOAP and BPEL4WS. It is also natural that this language has an XML notation, as this is true for the existing other Web service specifications. Although there is a need for a new XML specification for capturing QoS information, the concepts have already been well investigated in previous research efforts. The forthcoming OMG profile for QoS will hopefully become a successfully adopted standard for capturing QoS information in UML.

This work is founded by long-term research and practical experience. We therefore propose to define a transformation between this specification and an XML language with the same concepts and terminology, so that mainly the notation is different. Implementation of such a transformation is needed in order to fully automate some of the steps in our methodology. It is outside the scope of this paper to define the transformation.

**Pattern-wise Selection vs. Local and Global Selection.** A simulation tool has been realised, which benchmarks the pattern selection. Besides a local-greedy and the pattern selection, a third algorithm has also been implemented, which evaluates all possible assignments for all tasks within the composition. This selection, which is using a global perspective, always finds the best assignment considering the whole composition. But it scales exponentially with a growing number of tasks and candidates. The pattern selection can be seen as a specialisation of the global selection. The benefit is that it identifies the potential of QoS optimisation as explained in the example scenario. The algorithm performing a pattern-wise selection scales exponentially only with a growing number of candidates and tasks within a pattern element. Thus it will execute faster than a true global-based approach, if a composition consists of more than one pattern

element. The simulation tool has been realised in Java, J2SE 1.4. To ensure the optimal performance, the tool uses primitive data types and their arrays when performing the following steps:

1. Generation of the example structure based on the gas dispersion case.
2. Generation of candidates each with random QoS values for each task.
3. Performing each of the three selection methods each on the same composition with the same set of candidates.

The implementation of the tool, the random generation of the candidates and their QoS involve a large number of issues, which cannot be discussed in detail in this paper due to limited space. However, we would like to mention the following two main points:

- The simulation environment generates random QoS values for a given number of candidate services; however, for each task an optimal QoS is randomly set first and the values for the referring candidates are within zero and 100 percent worse compared to it. This ensures that generated QoS values remain in the same magnitude.
- For each setup with a particular number of candidates the simulation has been repeated for 25 times with different candidates. The results shown are the average over 25 repetitions for one setup.

Figure 8 shows a comparison of the resulting QoS by using the three different selection algorithms. The two upper lines show the ratio, how much better the QoS resulting from a global and pattern selection is compared to a local selection which is set to 1. The lower line shows how the global approach compares to the pattern approach. Figure 9 shows the computation times for the given setup. From these two Figures we can notice that the pattern approach shows that the increase of gained QoS compared to local selection is almost as large as with using the global selection. The computation time increases significantly when using pattern and global selection with an increasing number of candidate services. The local selection does not show a noticeable increase in computation time even with a relatively large number of candidates.

Comparing the absolute results, the global selection takes significantly longer than the pattern: e.g. looking at results for the setup with 60 candidate services, the pattern selection computes in about 13 seconds, while the global takes about 35 seconds. Also, for 90 candidate services the global selection takes about 2.5 times longer.

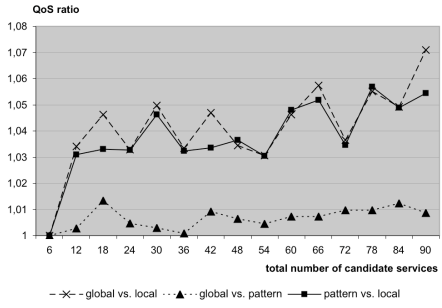


**Fig. 8.** QoS Ratio for Selection Methods

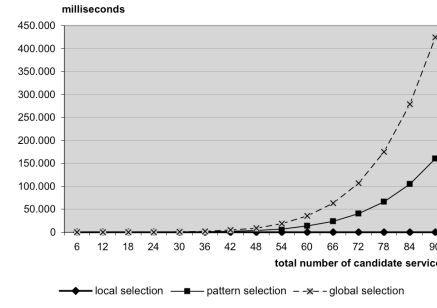


**Fig. 9.** Duration for Selection Methods

## 8 Related Work

The related work of QoS can be sorted in three categories: QoS modelling in UML, QoS Methodologies and QoS-optimisation approaches. In this section each of these fields is presented separately.

**QoS Modelling in UML.** OMG is in the process of finalising a standard UML profile for modelling QoS [9]. This UML profile defines a way to specify the QoS ontology with QoS characteristics such as we have used in Figure 4. It also has ways to express requirements with QoS requirements, which we have followed in Figure 3. The attachment of these QoS requirements to activities and activity models is an extension introduced in this report. Furthermore the concept of QoS interests is new in this report. Salazar-Zárate and Botella [10] have also proposed a way to use UML to capture similar concepts as OMG. NF-attributes correspond to QoS characteristics, NF-requirements correspond to QoS requirements, and NF-behaviour corresponds to QoS offered. In addition to OMG, they propose to use OCL expressions in a new class compartment to express constraints and inter-dependencies between the QoS model elements. Frølund and Koistinen defines an extension to UML called QML which is an alternative to OMG's forthcoming UML profile. QML also allows for definitions of QoS characteristics, QoS requirements and QoS offered (although with different terminology). QoS requirements can be associated with interfaces that are used by a requester, while QoS offered can be associated with interfaces that are implemented by a provider. OMG's UML profile uses stereotyped notes, while QML uses dashed rectangular boxes to define the QoS requirements and QoS offered. Our approach is more fine-grained by allowing to visually associate QoS requirements and QoS offered with an individual operation represented by a UML activity, while this is handled by textual scoping information in OMG's UML profile and QML. QML does not define a similar construction to our QoS interest.

**QoS Methodologies.** We have not addressed the reliability problem that advertised QoS values may not be accurate or up to date. Other work has addressed this issue basically in two ways. The first way is explained by Liu et. al [15] who suggest to use active monitoring and consumers feedback to ensure more correct QoS values of the provided services. The other way is to make a signed contract between the consumer and the service provider that ensures a certain QoS level. Contract negotiations and specifications are covered by Frølund and Koistinen in their QML [4]. Wang et al. [14] also deal with contract negotiation and argues that a mediating manager is needed to ensure the promised QoS of many concurrent clients. The execution time of a service may for instance depend on the number of concurrent clients. A mediating manager could handle admission control and resource management in order to ensure that the contracted clients get their entitled QoS, while others are blocked.

**QoS-Optimisation Approaches.** In our approach we have presented first the aggregation of QoS values and then the selection mechanism for choosing candidate services. This procedure is also found when looking at the evolution of related research. The foundation for the QoS aggregation in compositions can be seen by the work of Cardoso about calculating the QoS for workflows [2]. In his work he has identified different QoS characteristics and defined calculation rules. Because our composition patterns are based on the workflow patterns, we can also support structures like the two OR-split patterns along with the *m-out-of-n-join* constructs. As a consequence the composition

patterns can be easier applied to applications, which are derived from the workflow patterns by van der Aalst et al. [12]. Different authors have also discussed in various articles ([8], [1] or [16]), which QoS characteristics might be considered in compositions of Web services. Their contributions has been considered when determining the relevant QoS criteria for our methodology. If needed, the chosen QoS criteria can be easily extended with additional QoS criteria without affecting our methodology itself. For the selection of candidates, Zeng et al. have already identified a local and a global selection [16]. To address the problem that a global selection has unfeasible effort, they have introduced an Integer Programming (IP) solution to compute the optimal assignment of services, which reduces significantly the computation time according to their tests. We plan to test the IP-approach with our simulation tool to deliver a statement on how well it compares to pattern selection.

## 9 Conclusions and Future Work

We have introduced a UML model-driven methodology for designing and deploying Web service compositions, which uses a QoS-aware selection process for assigning Web service candidates to the tasks of a composition. OMG's forthcoming UML profile for modelling QoS can be used to capture most of the relevant QoS aspects in the different steps of our methodology, although we have proposed to extend it with the ability to specify QoS optimisation interests. Most of the steps in the methodology can be automated by existing or to-be-defined transformations.

We have also discussed different approaches for service selection and compared their characteristics. From our results we can see that different approaches for service selections would fit into different usage scenarios. The global approach optimises by taking all the different execution paths into consideration. It always performs the best QoS-optimised values but takes the longest time to compute. Thus it is not the best choice for run-time selection of services in time-critical applications or with complex compositions involving many tasks and candidate services. The local approach optimises each single task only, without considering the composition structure. It will always have the shortest computation time and has showed the lowest QoS-optimised values. The pattern approach optimises by taking the control flow patterns into consideration and results in almost the same QoS like the global selection while taking a shorter computation time. The summary is that all of the three approaches may be the best choice for a given usage scenario. With more simulation and testing it is possible to establish a run-time choice framework assuming all the three algorithms are available. It is important to consider if the selection process is carried out in run-time or design-time, and especially the users execution time requirement.

In summary we think that a model-driven methodology is the foundation for a tool-supported composition environment and thus along with the support for QoS the necessary contribution for designing compositions in industry applications. For future work we will adapt additional modelling methods besides the UML activity models for designing compositions. A planned enhancement for the selection of Web services is to add the support for semantic description of services as a selection criterion. This provides the opportunity for automated discovery of candidate services as well as creation

of the semantic descriptions for the new composition. Another future extension of our approach is to investigate how we can deal with QoS offerings of services that are dynamic with respect to the the number simultaneous clients and dynamic with respect to the input parameter contents.

**Acknowledgments.** The work of Roy Grønmo is partially funded by the European IST-FP6-004559 project SODIUM (Service Oriented Development In a Unified framework). We would also like to thank Gero Muehl, Gerhard Koehler and Jan Øyvind Aagedal for their valuable input.

## References

1. Benatallaah, Dumas, Fauvet, Rahbi, and Sheng. Towards Patterns of Web Services Composition. In *Patterns and Skeletons for Parallel and Distributed Computing*. Springer Verlag, 2002.
2. J. Cardoso. *Quality of Service and Semantic Composition of Workflows*. PhD thesis, Department of Computer Science, University of Georgia, Athens, GA (USA), 2002.
3. 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/, February 2005.
4. Svend Frølund and Jari Koistinen. Quality of Service Specification in Distributed Object Systems Design. In *Distributed Systems Engineering Journal*, 1998.
5. R. Grønmo, D. Skogan, I. Solheim, and J. Oldevik. Model-Driven Web Service Development. In *International Journal of Web Services Research (JWSR), vol. 1(4)*, October-December 2004.
6. Ching-Lai Hwang and K. Paul Yoon. Multiple Attribute Decision Making: Methods and Applications. In *Lecture Notes in Economics and Mathematical Systems, Vol. 186*. Springer Berlin, 1981.
7. M. C. Jaeger, G. Rojec-Goldman, and G. Muehl. QoS Aggregation for Web Service Composition using Workflow Patterns. In *Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC)*, Monterey, California, September 2004.
8. D. A. Menasce. Qos issues in web services. In *IEEE Internet Computing, vol. 6(6)*, pages 72–75, November-December 2002.
9. OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, August 2003.
10. G. Salazar-Zrate and P. Botella. Use of UML for modeling non-functional aspects. In *Proceedings of the 13th International Conference Software and Systems Engineering and their Applications*, Paris, France, December 2000.
11. W. M. P. v. d. Aalst. Don't go with the flow: Web Services composition standards exposed. In *IEEE Intelligent Systems, vol. 18(1)*, pages 72–76, 2003.
12. W. M. P. v. d. Aalst, A. H. M. t. Hofstede, B. Kiepuszewski, and A. Barros. Workflow Patterns. In *BETA Working Paper Series*, WP 47 2000.
13. I. Walsh, P. McCarthy, and J. Shields. e-Emergency Pilot Demonstrator, e-blana, Deliverable IST-2001-37724 ACE-GIS D1.2b, November 2003.
14. G. Wang, A. Chen, C. Wang, C. Fung, and S. Uczekaj. Integrated Quality of Service (QoS) Management in Service-Oriented Enterprise Architectures. In *Proceedings of the 8th International IEEE Enterprise Distributed Object Computing Conference (EDOC)*, Monterey, California, September 2004.
15. A. H. H. Ngu Y. Liu and L. Zeng. QoS Computation and Policing in Dynamic Web Service Selection. In *Proceedings of the Thirteenth International World Wide Web Conference*, New York, USA, May 2004.
16. L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalgnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. In *IEEE Transactions on Software Engineering, vol. 30(5)*, pages 311–327, May 2004.