Using Domain Ontology for Service Replacement Tasks: An Empirical Evaluation

Completed Research Paper

Paul Karaenke
Technical University of Munich
85748 Garching, Germany
karaenke@in.tum.de

Joerg Leukel
University of Hohenheim
70599 Stuttgart, Germany
joerg.leukel@uni-hohenheim.de

Vijayan Sugumaran
Oakland University
Rochester, MI 48309, USA
sugumara@oakland.edu

Abstract

Organizations increasingly use information technology (IT) to integrate their business processes into the processes of their suppliers, customers, and other third parties. An important IT approach is the realization of composite services that organize elementary software services under a shared workflow. Any failure of an elementary service can severely impact the process. The failed service must be examined and, ultimately, be replaced. In solving that task, the process designer must consider the quality-of-service (QoS) of the process. However, the heterogeneity of service descriptions puts the burden on the designer. This research empirically evaluates how designers can use a domain ontology, namely the QoS aggregation ontology, for the replacement task. We report on a laboratory experiment to compare the effectiveness and efficiency of using the ontology vis-à-vis an aggregation table. The results provide evidence for the usefulness of the domain ontology that specifies problem-solving knowledge required for a time-critical task.

Keywords: Design science, knowledge-based systems, laboratory experiment, service composition, service quality

Introduction

Services computing relies upon software services that can be used for realizing business processes "as a service", i.e., by arranging a number of elementary services into a composite service (Zhao et al. 2007). This idea is important to businesses because a composite service may include services from multiple service providers but still ensure a certain behavior (Iyer et al. 2003). Empirical research provides evidence that services computing improves the performance of interorganizational business processes (Becker et al. 2011; Krishnan et al. 2007; Oh et al. 2007). Services can only implement a business process if they sufficiently meet the requirements of that process (Basu and Kumar 2002). Therefore, in case of service failure, the process designer must decide whether alternative services fit into the process and replace the faulty service at process runtime. This task is called *service replacement*.

With increasing number of software services offered via standardized protocols, service replacement gains greater importance for many organizations. This increase has been amplified by the proliferation of standards such as SOAP, WSDL, and WS-Policy by the World Wide Web Consortium (these services are also referred to as web services). Although exact numbers are not available, it is reasonable to assume that

tens of thousands of web services are available. For instance, the online portal *ProgrammableWeb*[1] lists more than 15,000 open application programming interfaces (API) for developing web services. Large catalogues of web services for different domains exist, e.g., life sciences (Bhagat et al. 2010) and geography (Friedrich Schiller University Jena 2015).

Service replacement is non-trivial because of two reasons. First, the alternative service has to guarantee a certain threshold for quality-of-service (QoS) parameters such as availability and throughput (O'Sullivan et al. 2002). Second, the position of the service within the execution sequence has to be considered. The latter is required for calculating the QoS of the composite service, which is an aggregation of the QoS of all elementary services. While prior research provides various methods for replacing services (e.g., Canfora et al. 2008; Liu et al. 2010), these methods aim for full automation of the replacement task. Automation is only possible by avoiding the difficulties incurred by the *heterogeneity of QoS parameters*. This assumption may not hold in an inter-organizational setting because service providers rarely subscribe to a shared vocabulary for service description. As shown in a recent survey, the models for representing QoS vary a lot in expressiveness, level of detail, and domain dependency (Kritikos et al. 2013). Therefore, process designers are confronted with varying representations of similar, though potentially distinct QoS parameters. The designer must map corresponding parameters onto each other, and then determine for each parameter pair the aggregation formula. Solving this task requires knowledge of the correct aggregation formulae for combinations of so called composition patterns and particular types of parameters (Jaeger et al. 2004).

Prior research provides two distinct approaches to represent the problem-solving knowledge. The most used approach relies upon *QoS aggregation tables*, which define for each combination of composition pattern and parameter type the correct aggregation formula (e.g., Alrifai et al. 2012; Fakhfakh et al. 2013; Sun and Zhao 2012). An aggregation table is complex and the designer's ability to use it correctly might be undermined by the heterogeneity of QoS parameters. Negative effects of wrongly aggregating the QoS parameters of constituent services can lead to an overestimation of the QoS for the composite service. For example, when taking the maximum instead of the product for the availability of multiple services executed in sequence, the resulting availability of the composite service would be overstated. When guaranteeing this availability to clients or using it in risk assessment, unexpected cost can arise for the company executing the process.

In our prior research, we proposed the *QoS aggregation ontology* as an alternative approach (Karaenke et al. 2013). This ontology formalizes parts of the problem-solving knowledge into a domain ontology that could be used by a human process designer (Roa et al. 2014). The ontology will be provided to the designer as a diagram. However, it is still not known whether process designers using the QoS aggregation ontology will arrive at better task solutions than when using a QoS aggregation table. Empirical evaluations of either approach have not been carried out yet. We fill this gap in the literature by providing an empirical evaluation of the QoS aggregation ontology. We designed and performed a laboratory experiment involving novice process designers. The experiment had appropriate sample size to ensure that our statistical analysis would have adequate power. Our research provides important insights into the usefulness of an ontology-based problem-solving approach and advances our prior design science research (DSR) through a rigorous empirical evaluation (Hevner et al. 2004). The contribution of this research is the empirical evaluation (evaluate process of DSR), while our prior research developed the ontology artifact and provided a preliminary evaluation using a particular scenario and simulation (Karaenke et al. 2013).

The remainder of this paper is organized as follows. We first discuss the background to our research, followed by the presentation of our hypotheses. Then we describe the experiment we conducted to test these hypotheses and report on the results. This is followed by a discussion of the implications and limitations of our research, before concluding the paper.

[1] http://www.programmableweb.com/

Background

We first define the service replacement task and then discuss problem-solving approaches from the perspective of a human process designer.

Service Replacement Task

Service replacement is a task that is carried out if a service within a composite service has to be substituted by another service. There might be various reasons for this task substitution such as technical outage, long response times, and unexpected functional behavior (Gold et al. 2004). These reasons can be subsumed under situations in which the current service fails to meet the requirements imposed by the composite service. In the following, we denote the to-be-replaced service as *faulty service*. The faulty service must be replaced by a *candidate service*. The set of candidate services is usually demined by a service discovery component, which identifies only candidate services that meet the functional requirements.

Let us consider a composite service that extracts data, followed by several transformation steps, and finally loads the data into a data warehouse. The workflow for that service is shown in Figure 1. This workflow contains six tasks (represented by rectangles) and four gateways (represented by diamond shapes). Tasks are the activities, which will be carried out by services. Thus, prior to workflow execution, services must be bound to tasks. Let us further consider that the services bound to the tasks shown in Figure 1 are described by two QoS parameters: cost and execution time. Calculating the QoS of the composite service will be different for the two parameters and contingent upon the execution order of the services involved. For instance, because the services for "Classify data" or "Annotate data" will be executed in parallel, the QoS of this workflow fragment is the sum of the two cost values and the maximum value of the two execution time values.

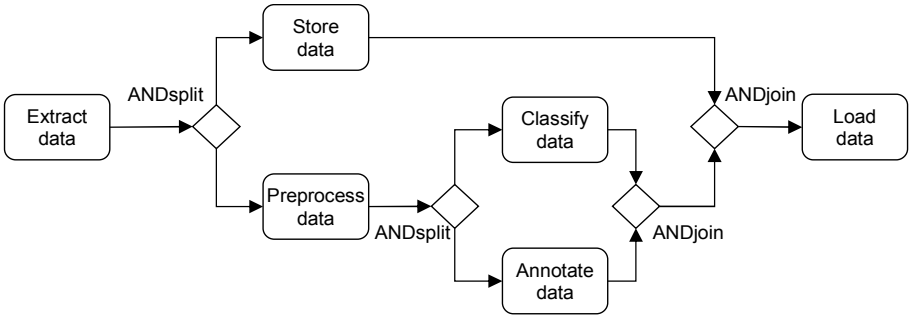


Figure 1. Example workflow

To assess the suitability of the candidate services, the resulting composite QoS has to be calculated, which requires understanding the semantics of QoS parameters. Therefore, the overall procedure can be structured into three steps: First, corresponding parameters of the faulty and candidate service must be mapped onto each other. Second, QoS aggregation formulae for each parameter pair must be determined. Third, the QoS of the composite service with the candidate service must be calculated.

The literature provides a rich body of methods for automating the service replacement task by assuming *homogeneous* QoS parameters. Under this assumption, the mapping of corresponding parameters of the faulty and candidate service is trivial. Thus, services computing research focuses on the QoS aggregation subtask (i.e., the two latter steps of the service replacement task). QoS aggregation is nowadays well understood because it is essential to QoS-aware service composition (Blake et al. 2012). While aggregation formulae for specific QoS parameters have been proposed, prior research also stresses that these parameter sets can be extended retaining the overall aggregation approach (Ardagna and Mirandola 2010; Fakhfakh et al. 2013; Zeng et al. 2004). Abstracting from specific QoS parameters is thus important for a domain-independent approach to QoS aggregation. Similarly, control flows can be abstracted by *composition patterns* (Jaeger et al. 2004), which are based on workflow patterns (van der Aalst et al.

2003) and represent basic structural elements of service compositions. The respective aggregation formulae can then be determined with these patterns for different QoS parameters.

While the range and scope of both composition patterns and QoS parameters considered in the literature differ, the common approach relies on graph reduction algorithms (Alrifai et al. 2012; Ardagna et al. 2010; Cardoso et al. 2004). These composition patterns and formulae have been adopted and refined, for example, in service selection at the process level (Ardagna et al. 2010), service selection and late binding during process execution (Huang et al. 2009; Liu et al. 2012), and service replacement upon actual constituent service failure during execution (Canfora et al. 2008; Li et al. 2011; Lin et al. 2010). In addition, specific formulae for predicting the QoS based on probabilistic functions have been proposed (Hwang et al. 2007; Stein et al. 2009). *Execution routes* are an alternative approach to the composition pattern-based aggregation (Yu e tal. 2007). An execution route is built for each possible execution path, resulting in a set of deterministic alternatives. While this approach reduces the effort for determining the aggregation formulae and calculating the composite QoS, it requires the expansion of workflows to all the deterministic sequential paths, which is computationally demanding.

Heterogeneity of QoS parameters may have at least three manifestations: (1) The parameter names used can be different, (2) parameters can take different positions in the list of parameters, and (3) the number of parameters can be different. In this case, understanding the semantics of QoS parameters is essential to arrive at a correct mapping of corresponding parameters. The naïve solution is to state the aggregation formulae for each QoS parameter explicitly in service descriptions (Haq et al. 2011). However, this solution is not feasible because it requires all service providers to determine the correct formulae for all of their QoS parameters considering all potential usages of their services in workflows. The mapping problem for QoS parameters could be mitigated if service providers amend their service descriptions with semantic annotations conforming to a shared conceptualization of QoS parameters, for which several QoS ontologies have been proposed (Dobson et al. 2005; Muñoz Frutos et al. 2009; Tran et al. 2009). This approach, however, assumes that (1) *all* service providers subscribe to that ontology, and (2) the annotations are available *prior* to service runtime; both assumptions are not realistic in an inter-organizational setting. In summary, service replacement tasks for heterogeneous QoS parameters still require the involvement of a human process designer. Then, the question is how to represent the relevant problem-solving knowledge to the designer. Next, we discuss two representations and problem-solving approaches proposed in the literature.

Problem-Solving Approaches

The two problem-solving approaches that we discuss have in common that they rely upon composition patterns for QoS aggregation (Jaeger et al. 2004). Composition patterns have been proposed long ago and their effectiveness in solving QoS aggregation problems has been shown (Alrifai et al. 2012; Canfora et al. 2008; Hwang et al. 2007; Zheng et al. 2013). We consider seven composition patterns that reflect a wide array of possible workflows: *Sequence*, *Loop*, *XORXOR*, *ANDAND*, *ANDDISC*, *OROR*, and *ORDISC* (Jaeger et al. 2004). For example, the *ANDAND* pattern denotes an *AND-split* gateway followed by an *AND-join* gateway. When *AND-split* and *OR-split* gateways are followed by the *m-out-of-n-join*, this is denoted by the *ANDDISC* and *ORDISC* patterns, respectively. The *m-out-of-n-join* is used to continue execution after m out of n preceding branches have been completed; e.g., to increase availability, multiple equivalent services are invoked in parallel while only one of the responses is processed.

First is the QoS aggregation table, which is a compact representation of the many dependencies between composition patterns and parameter types from which the designer can directly select the aggregation formulae. Second is the QoS aggregation ontology (Karaenke et al. 2013), which enables the process designer to add semantic annotations to QoS parameters to infer the correct aggregation formulae.

QoS Aggregation Table

The QoS aggregation table is a tool that can be used to determine the correct aggregation formula for a given QoS parameter under consideration of the position of the service within the execution sequence. Thus, the aggregation formula is contingent upon two factors, parameter type and composition pattern. If the process designer knows the type to which a given QoS parameter belongs to as well as the composition pattern for the service under investigation, the designer can retrieve the correct formula from that table.

Once this retrieval has been completed, the final step of the service replacement task, i.e., calculating the composite QoS, can be performed (usually by an IT system, not the human problem-solver).

With respect to the parameter type factor, we note that a variety of QoS aggregation tables have been used in prior research. While some works are limited to concrete parameters (e.g., Canfora et al. 2008; Mukherjee et al. 2008; Liu et al. 2012), other works have identified parameter types, which contain multiple parameters that share the same aggregation formula (e.g., Fakhfakh et al. 2013; Hwang et al. 2007; Sun and Zhao 2012). Thus, an important finding is the generalization from concrete parameters to parameter types specifically for QoS aggregation. Let us consider two services that are described by two parameters, *cost* and *execution time*, and are executed in parallel. Aggregating their QoS into a composite QoS is different for *cost* (i.e., sum of the two values) and *execution time* (i.e., maximum value of the two). Therefore, the two parameters belong to different parameter types. The rationale is that parameters sharing the aggregation formulae for all composition patterns also share the parameter type. That is, parameter properties that are irrelevant for QoS aggregation are not considered in the parameter types.

Prior research has identified these parameter types (Sun and Zhao 2012; Karaenke et al. 2013), for which we provide definitions as follows:

- Type 1 parameters are aggregated by summation in sequential and parallel executions (e.g., cost).
- Type 2 parameters are aggregated by summation in sequential executions and by determining the maximum in parallel branches (e.g., response time).
- Type 3 parameters are aggregated by determining the minimum in sequential executions and by summation for the upper bound in patterns starting with an OR-split (e.g., bandwidth).
- Type 4 parameters are aggregated by multiplication in sequential and parallel executions (e.g., availability).
- Type 5 parameters are aggregated by determining the minimum in sequential executions and by determining the maximum for the upper bound in patterns starting with an OR-split (e.g., encryption).

Both factors – composition pattern and parameter type – span a matrix of 7*5=35 cases, with each cell giving the respective aggregation formula. Note that such formulae can only be applied by assuming that related parameters have the identical unit of measurement (UoM). The problems imposed by heterogeneous UoM are not specific to QoS and solutions have been proposed (e.g., Beaty 2006). Finally, we must consider non-deterministic workflows that are due to particular composition patterns, i.e., *XORXOR*, *ANDDISC*, *ORDISC*, and *ORDISC*. For instance, in case of a *XORXOR* pattern, we do not know a priori which task following the *XOR-split* will be activated, thus which service will be executed. Therefore, in aggregating the QoS parameters, either option must be accounted for by distinguishing the *upper* and *lower bound* of the QoS. In summary, the initial table expands to a 7x5x2-matrix as shown in Table 1, which is an adaption of the table used by Karaenke et al. (2013).

Table 1. QoS aggregation table

Type	Bound	Sequence	Loop	XORXOR	ANDAND	ANDDISC	ORDISC	ORDISC
1	Upper	*sum*	*linear*	*max*	*sum*	*sum*	*sum*	*sum*
	Lower	*sum*	*linear*	*min*	*sum*	*sum*	*min*	*min*
2	Upper	*sum*	*linear*	*max*	*max*	*max*	*max*	*max*
	Lower	*sum*	*linear*	*min*	*max*	*min*	*min*	*min*
3	Upper	*min*	*identity*	*max*	*min*	*min*	*sum*	*sum*
	Lower	*min*	*identity*	*min*	*min*	*min*	*min*	*min*
4	Upper	*product*	*power*	*max*	*product*	*product*	*max*	*max*
	Lower	*product*	*power*	*min*	*product*	*product*	*product*	*product*
5	Upper	*min*	*identity*	*max*	*min*	*min*	*max*	*max*
	Lower	*min*	*identity*	*min*	*min*	*min*	*min*	*min*

A process designer can use the QoS aggregation table for service replacement tasks in the following way: First, the designer will map corresponding parameters of the faulty and candidate service onto each other. Second, the designer will consult the table to determine the aggregation formula for the candidate service's parameter by identifying the parameter type to which the parameter belongs to (table row) and selecting the current composition (table column); the latter is usually available from the workflow and thus no decision needs to be made. Once this has been completed for each parameter of the candidate service, the third step of the replacement task (mathematical aggregation) can start with no human intervention. Thus, the QoS aggregation table supports the designer in the second step of the service replacement task (formula determination) but not in the first step (parameter mapping).

QoS Aggregation Ontology

The QoS aggregation ontology is a formal specification of the dependencies between composition patterns, parameter types, bounds, and aggregation formulae through classes and relationships using a highly expressive modeling grammar. The particular grammar used is Web Ontology Language (OWL) (W3C 2004), specifically its sublanguage OWL DL, which provides the required expressiveness because of description logics (DL) as the underlying knowledge representation mechanism (Baader et al. 2010). OWL DL retains computational completeness and decidability unlike the sublanguage OWL Full, whereas the sublanguage OWL Lite lacks expressiveness (Horrocks et al. 2003). This domain ontology can be used to annotate QoS parameters, i.e., by defining links between parameters and ontology classes; hence, parameters become instances (individuals) of the knowledge base. Then, the knowledge representation mechanism allows a reasoner to infer the correct aggregation formulae based on the annotations of QoS parameters provided by the human designer.

Figure 2 shows a diagrammatic overview of the ontology. This figure is an adaptation of the original diagram used by Karaenke et al. (2013), e.g., some abbreviations have been spelled out. The ontology provides taxonomies for (1) composition patterns, (2) parameters (divided into parameter types and bounds), and (3) aggregation formulae, which are linked through associations. The diagram can be read as follows: An instance of *AggregationFormula* is defined for one or more instances of *Parameter* and for one or more instances of *CompositionPattern*.

Figure 2 does not show the 70 combinations that are also formally specified in the ontology. The many combinations are represented as DL axioms. These axioms are restrictions over the *forP* and *forCP* associations (so called role restrictions in OWL DL). For example, because a sequence of *Type4* parameters must be aggregated by the *product* formula, the definition of the class *Product* includes restrictions ($\exists$, $\forall$) as shown in the following axiom: $Product \equiv \exists forP.Type4 \sqcap \forall forCP.Sequence$.

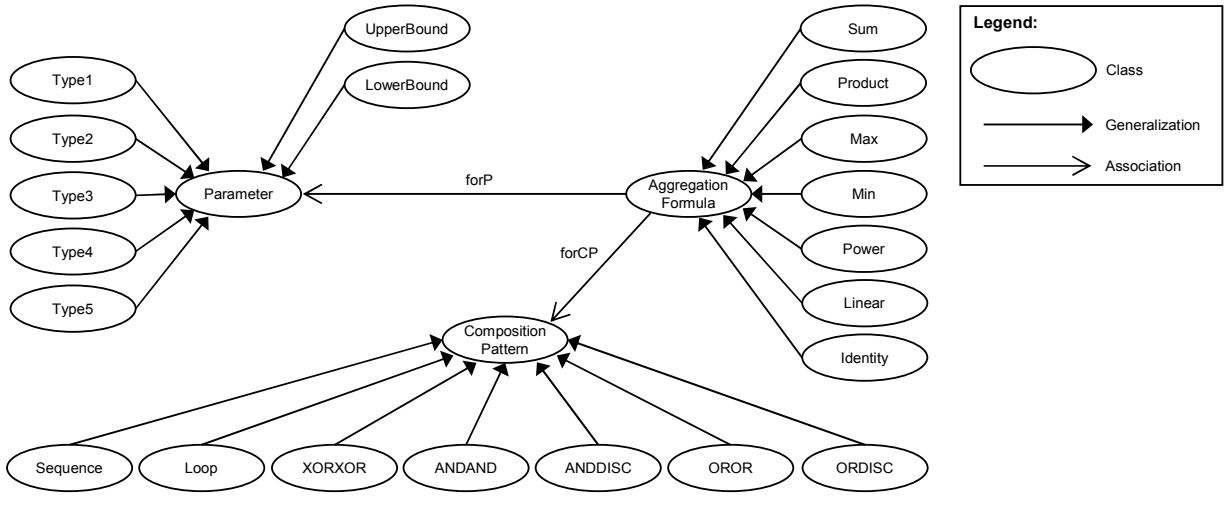


Figure 2. Diagrammatic representation of the QoS aggregation ontology

While the original proposal of the QoS aggregation ontology focused on the formal representation and the use of this representation for computing the composite QoS, its use for service replacement tasks in the organizational context of a process designer has not been studied. Specifically, we argue that the ontology can assist designers in the first step of service replacement, i.e., in the mapping of corresponding parameters. We discuss this assistance for the scenario shown in Figure 3.

As Figure 3 shows, one faulty and one candidate service are each described by two QoS parameters, which have different names but are semantically equivalent, and the services are part of an *ORORO* composition pattern. To calculate the composite QoS when replacing the faulty service with the candidate service, the process designer would usually need to find out the mapping of their parameters onto each other and the right aggregation formulae. However, the designer can use the ontology, of which only the taxonomy of QoS parameters will be shown to the designer. This taxonomy comprises the five parameter types and several subclasses of these types to account for possible parameters that have different semantics but share the same aggregation formulae. For instance, Figure 3 shows that the *Type2* class has three subclasses for response time, execution time, and downtime. While each class represents time-related QoS, their semantics is different and must be considered separately. Similarly, the *Type3* class has two subclasses for bandwidth and storage capacity. To not overload the diagram for the purpose of our demonstration, we have included only one subclass for the Type1 class and omitted all other types (note that we used a larger taxonomy in our experimental evaluation).

Using the ontology, the problem-solving proceeds as follows:

1. The designer annotates each QoS parameter by linking it to one parameter class. For instance, the faulty service's parameter "Throughput" is linked with the "Bandwidth" class. The designer can make these annotations independently by examining each service description. However, if both descriptions are visible at the same time, designers may add annotations in a sequence that best meets their personal preference or style.
2. The designer selects the composition pattern for the services in the process. Because the pattern is available from the workflow, this selection can easily be automated.
3. The system that implements the ontology infers the aggregation formulae (upper and lower bounds) for each QoS parameter through DL reasoning.

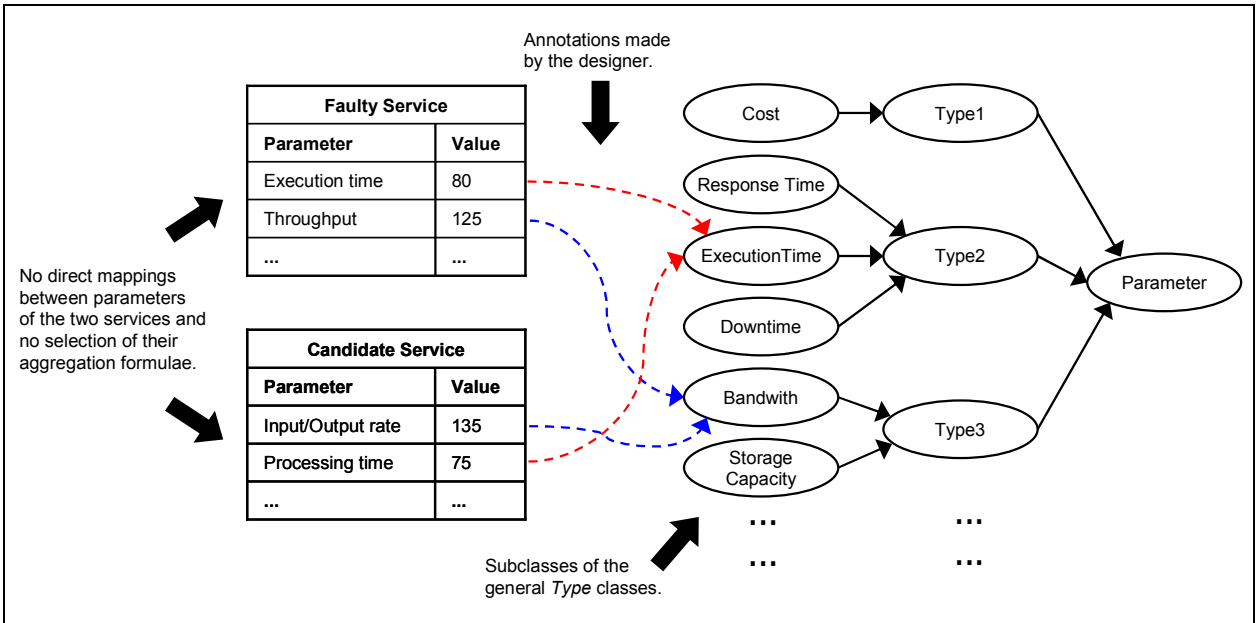


Figure 3. Problem-solving using the QoS aggregation ontology

Next, we depict how DL reasoning allows making the inferences for the "Throughput" parameter and its upper bound formula. First, the system will be initialized through (1) adding three individuals to the knowledge base, namely one each for *CompositionPattern*, *UpperBound*, and *AggregationFormula*, and (2) defining associations between these individuals. We can define the initialization as follows:

{*myCP*}:*CompositionPattern*; {*myP*}:*UpperBound*; {*myAF*}:*AggregationFormula*

({*myAF*}, {*myCP*}):*forCP*; ({*myAF*}, {*myP*}):*forP*

The ultimate goal is to derive to which subclass of *AggregationFormula* the individual *myAF* belongs to. As a result of step 1 of the problem-solving procedure described above, the individual *myP* will also become an instance of the *Bandwith* class, i.e., {*myP*}:*Bandwith*. The second step assigns the individual *myCP* to the *ANDAND* class, i.e., {*myCP*}:*ANDAND*. In the third step, the system is queried for all the classes to which the individual *myAF* belongs to. The system will return the inferred statement {*myAF*}:*Sum* because the individual *myAF* fulfills the restrictions that define the *Sum* class. Specifically, the DL fragment that allows this inference is as follows:

$\exists forP.Type3 \sqcap \forall forCP.OROR \sqcap \forall forP.UpperBound$

In summary, the main difference between using the ontology and using the QoS aggregation table is that the designer neither determines the intricate combination of parameter type, composition pattern, and bound nor directly maps heterogeneous QoS parameters onto each other. The mapping of the parameters onto the vocabulary provided by the subclasses of the ontology is done by the designer. The performance of the designer will then dependent on their ability to understand the semantics of these classes.

Hypotheses

We hypothesize that using the QoS aggregation ontology has positive impact on (1) problem-solving performance, (2) task completion time, and (3) perceived ease-of-use. The basis for our hypotheses is as follows: Problem-solving is simplified by adding semantic annotations to each parameter of the two services. In making the annotations, the designer maps QoS parameters onto a small set of parameter classes defined in the ontology. Understanding the semantics of the parameter classes will likely be easy, and the mapping will always be done onto the same target conceptualization (the ontology). In deciding about these annotations, the designer will not require specific knowledge about the dependency of aggregation formula on composition pattern and parameter type but need to understand the parameter names used. In other words, the service replacement task becomes a classification task, i.e., deciding to which (parameter) class a (parameter) individual belongs to. Then, the system that implements the ontology will infer the correct formulae based on the annotations and the composition pattern.

The use of parameter classes in the problem-solving process outlined above resembles a phenomenon of human information processing: If we know very little about an individual except that it belongs to a particular concept, then we can infer all or many of the concept's properties for the individual (Smith and Medin 1981). This phenomenon is known as the *categorization function of concepts*. Similarly, if the designer does not know the aggregation formulae for a particular parameter (individual) of a specific set of services in a workflow, then from knowing the parameter class to which the parameter belongs to, we can infer the missing information. We believe that this resemblance of the problem-solving process provides a theoretically sound argument that the QoS aggregation ontology will effectively support human problem-solvers in service replacement tasks.

With no ontology, however, the designer has to define mappings from a specific source conceptualization (valid only for the candidate service) onto a specific target conceptualization (valid only for the faulty service), and use the aggregation table to determine the formulae. The designer would define direct mappings between corresponding parameters of the faulty and candidate service. For the example shown in Figure 3, the designer would first map the *Throughput* parameter of the faulty service onto the *Input/Output rate* parameter of the candidate service. Then, the designer would assign a formula to that parameter by considering the composition pattern and the parameter type.

H1: *Process designers using the QoS aggregation ontology will perform better in service replacement tasks than those using the QoS aggregation table.*

We further expect that the ontology will enable the designer to arrive at the solution more quickly. The rationale is that the problem-solving process might require less mental effort. The designer can concentrate on constructing mappings onto the uniform target conceptualization, for which the cognitive processes might exhibit less variance.

H2: *Process designers using the QoS aggregation ontology will require less time for service replacement tasks than those using the QoS aggregation table.*

In addition, we anticipate that designers will become familiar with the rather small-scale ontology and experience comfort in relating QoS parameters to parameter classes. That is, the perceived mental resources expended might be lower compared to situations where links between variant conceptualization must be established. Ease-of-use is a proxy for the overall effort required for the task solution.

H3: *Process designers using the QoS aggregation ontology for service replacement tasks will perceive higher ease-of-use than those using the QoS aggregation table.*

Method

To test our hypotheses, we conducted a laboratory experiment for which we report on the experimental design and data collection in this section.

Experimental Design

In the experiment, participants were asked to solve replacements tasks using the QoS aggregation ontology or the aggregation table. We developed a web application based on the *oTree* framework (Chen et al. 2016) that enabled participants to complete most parts of the experiment in that application. The factor under investigation was the use of the QoS aggregation ontology. Thus, our independent variable had two levels, ontology vis-à-vis table. Because the problem-solving processes for these two levels were distinct, we used a repeated measures design, where all participants were exposed to both conditions of the independent variable. This design allowed us to effectively control for individual differences such as prior knowledge and motivation. We controlled carryover effects by using a counterbalanced design with two separate groups of participants (i.e., participants in group 1 received the conditions in the order ontology-table, whereas participants in group 2 received the conditions in the order table-ontology).

Participants

The experiment was targeted at persons who possess basic knowledge of business process modeling and service description. The participants should have an understanding of how the aggregation of QoS parameters is affected by the composition pattern and the particular parameter used. This understanding does not depend on any specific modeling grammar or technical specification for software services. Therefore, the target group for the generalization of our findings is novice process designers who are involved in designing and maintaining composite services that realize business processes.

To determine the required sample size, we ran a pilot study with nine participants (eight PhD students, one undergraduate student). Participants achieved higher scores in problem-solving when using the ontology (M=59.56, SD=2.70) than when using the aggregation table (M=54.22, SD=5.59), suggesting a large size effect (Cohen 1988). We performed an a priori power analysis using the *G*Power 3.1* tool (Faul et al. 2007). The analysis (Cohen's d=0.8, p=.05, Wilcoxon-signed rank test, one-tailed) revealed that we must have at least 20 participants providing 40 observations to achieve a high statistical power (P=.95). Therefore, we recruited 26 students for the main study. This sample size is large enough to detect large size effects for our repeated measures design.

Our participants were students (22 male, 4 female) from a business school of a university in Western Europe. 18 participants were enrolled in an undergraduate information systems (IS) program, 3 in a graduate IS program, and 5 in a graduate management program with IS major. Due to attending prior compulsory courses in IS and business management, all participants possessed the knowledge and skills required for the experiment. Participation was voluntary and the participants were awarded 10 Euros. To further motivate students, an additional compensation of 5 Euros was paid if at least one-half of the aggregation formulae were correct.

Measurements

Consistent with our hypotheses, we used three dependent variables. *Problem-solving performance (PSP)* was defined as the absolute number of correct aggregation formulae for the QoS parameters. For instance, if the faulty service had five parameters and the candidate service provided five corresponding parameters, then a maximum score of 5*2=10 could be obtained (because of the uppler/lower bound formulae for each parameter). No coding was necessary because participants used computer terminals. All user inputs were stored, and the problem-solving scores were computed automatically. Similarly, the *completion time* was measured for each task, recorded, and aggregated into total time. *Perceived ease-of-use (PEOU)* was measured at the end of each condition. A three-item scale was adapted from technology acceptance research (Moore and Benbasat 1991), and tailored to the terminology of our task setting (materials provided in Appendix A3).

We used control variables to account for personal factors. Prior research suggests that using domain representations (e.g., process models, conceptual schemas) for problem-solving might be contingent upon modeling knowledge and domain knowledge (Khatri et al. 2006; Mendling et al. 2012; Reijers and Mendling 2012). Because solving our experimental tasks did not require knowledge of a particular modeling grammar, we only used a perceptual measure for self-reported process modeling knowledge *(Kn-PRO)* by adopting a three-item instrument (Mendling et al. 2010) (item definitions provided in Appendix A1). In addition, we used age and credits as measures for study progress. As students progress through the university they acquire knowledge of both the IS field and application domains. While we did not expect effects of age and credits, we anticipated that the participants in our sample differed in both variables (considering that our sample included undergraduate and graduate students).

Materials

The experimental materials were designed to fulfill two requirements. First, the tasks covered all the manifestations of service heterogeneity. The first group of tasks provided candidate services with parameters that differed in name and order. In the second group of tasks, the parameter names were different, the parameters took different positions, and some parameters were excess parameters that had no equivalent in the parameter list of the faulty service. We then varied the number of parameters per service. We defined tasks with 5 and 7 parameters for the faulty services, and 5, 7, and 9 parameters for the candidate services. In total, we defined 6 tasks (12 representations because of two conditions).

Second, the parameter names were representative for descriptions of software services. We first identified a set of commonly used parameters from extant literature (e.g., Alrifai et al. 2012; Fakhfakh et al. 2013; Liu et al. 2012). Then, we retrieved synonyms from the *WordNet* lexical database (Princeton University 2016). The final set of parameters included 33 names that were grouped into nine synsets as shown in Table 2 (with each synset providing words that are interchangeable). Based on these synsets, we randomly assigned parameters to faulty and candidate services to create service descriptions of considerable heterogeneity.

Table 2. Parameters used in the experiment (per parameter type)

Type	Parameter class	Synset
1	Cost	Cost, Cost per Month, Fee, Monetary Value
2	Response Time	Latency, Reaction Time, Response Time
	Execution Time	Cycle Time, Execution Time, Processing Duration, Processing Time
	Downtime	Downtime, Service Outage, Service Outage Period, Service Outage Time
3	Bandwidth	Bandwidth, Data Transmission Rate, Input/Output Rate, Throughput
	Storage Capacity	Capacity, Storage Capacity, Storage Volume
4	Availability	Accessibility, Availability, Dependability, Reliability, Service Availability
	Accuracy	Accuracy, Correctness, Exactness
5	Encryption	Data Encryption, Encryption, Encryption Key Length

Procedures

The experiment was structured as follows: In the *preparation phase*, the experimenter explained the general procedures, followed by a brief presentation of the service replacement task in services computing. The presentation explained the dependency of aggregation formulae on the combination of composition pattern and parameter type. To avoid experimenter bias, this presentation was free from any hints on *how* to solve the replacement task. Any information on the two problem-solving approaches was provided by the web application only.

After filling out a web questionnaire on demographic data and prior process modeling knowledge (materials provided in Appendix A1), participants were randomly assigned to one of the two treatment conditions to start the *training phase*. In this phase, the application displayed a tutorial page describing one of the problem-solving approaches. Participants were given ample time to read the tutorial page until they pressed the next page button. Then, an example task was presented. The candidate and faulty service had each three parameters, thus solving this task was likely easy. The participants were asked to provide their solution. The system showed whether the solution provided was correct or incorrect. In the latter case, the participants had to revise their solution. This step was repeated until the correct solution was entered.

Then, the *experimental phase* began. The application presented a service replacement task. Example screenshots for each problem-solving approach are provided in Appendix A2. After the solution was entered, the application moved on to the next task. When the final task of the condition had been completed, the participants answered the PEOU questions. This procedure was repeated for the second problem-solving approach (sequence: tutorial page, example task, actual tasks, and PEOU questions). To mitigate any learning effects, the order of tasks was randomized.

Results

We first examine the conformance of the data with the assumptions of statistical tests. Then, we present the results from testing our hypotheses.

Data Screening

Table 3 shows descriptive statistics. On average, participants reported a medium level of process modeling knowledge (Kn-PRO). With respect to our dependent variables, participants achieved medium scores in problem-solving when using the table. When using the ontology, participants achieved higher scores, required about half of the time, and reported slightly higher ease-of-use. Overall, the descriptive statistics were in line with our expectations.

Table 3. Descriptive statistics (N=26)

Type of variable	Variable	Scale	Min	Max	M	Mdn	SD
Control variables and manipulation checks	Age	Years	20	28	23.27	23.00	1.18
	Credits	0-300	35	282	146.15	148.50	51.65
	Kn-PRO	1-7	1	6.33	3.73	4.00	1.66
Dependent variables	PSP-T	0-80	4	80	48.46	57.00	24.07
	PSP-O	0-80	32	80	69.46	74.00	11.34
	Time-T	Sec.	544.49	2156.63	1132.61	1085.00	354.64
	Time-O	Sec.	299.21	1082.86	574.05	626.10	214.06
	PEOU-T	1-7	1	6.33	3.63	3.67	1.53
	PEOU-O	1-7	1	7.00	4.09	3.83	1.64

Variables: ***Kn-PRO***: Self-reported process modeling knowledge; ***PSP-T/O:*** Problem-solving performance using the table or ontology; ***Time-T/O***: Time required using the table or ontology; ***PEOU-T/O***: Perceived ease-of-use using the table or ontology.

We examined the reliability of the scales for self-reported process modeling knowledge (three items), problem solving (six items) and perceived ease-of-use (three items). Cronbach's alpha was .956 for Kn-

PRO, .964 for PSP-T, .948 for PSP-O, .837 for PEOU-T., and .865 for PEOU-O, which indicate sufficient reliability for each scale. An examination for normal distribution found departures for the performance variables. We decided to run non-parametric tests, which have also higher power compared to t-tests considering our sample size.

In the next step of our analysis, we assessed the correlations between our independent and dependent variables (results shown in Table 4). Two results are noteworthy. First, self-reported process modeling knowledge was the only control variable positively correlated with problem-solving performance (when using the ontology). Second, those who performed better also experienced higher ease-of-use (and spent less time when using the ontology).

Table 4. Correlation matrix

Variable	Age	Credits	Kn-PRO	PSP-T	PSP-O	Time-T	Time-O	PEOU-T	PEOU-O
Age	1								
Credits	.199	1							
Kn-PRO	-.083	-.138	1						
PSP-T	-.087	.030	.348	1					
PSP-O	-.115	-.081	.566**	.483*	1				
Time-T	.481*	-.084	-.263	-.155	-.266	1			
Time-O	.174	.237	-.484*	-.127	-.459*	.127	1		
PEOU-T	-.091	-.218	.182	.609**	.185	-.107	-.189	1	
PEOU-O	-.144	-.096	.714**	.329	.668**	-.349	-.681**	.226	1

*p<.05; **p<.01 (Spearman's Rank Correlation Coefficients, 2-tailed).

Variables: ***Kn-PRO***: Self-reported process modeling knowledge; ***PSP-T/O:*** Problem-solving performance using the table or ontology; ***Time-T/O***: Time required using the table or ontology; ***PEOU-T/O***: Perceived ease-of-use using the table or ontology.

Hypotheses Testing

Table 5 provides the results of our hypotheses testing. We found strong support for H1 and H2; hence using the ontology improves problem-solving performance and reduces the time required compared to using the table. Both effects were of large size. However, the test of H3 indicated a nonsignificant effect on the PEOU (thus no support for H3). Power analysis revealed that in order for an effect of this small size (d=0.3) to be detected (95% chance) as significant at the 5% level, a sample of 128 participants (256 observations) would be required.

Table 5. Results of hypotheses testing

Condition		Table			Ontology			Test		
Variable	**Scale**	**M**	**Mdn**	**SD**	**M**	**Mdn**	**SD**	**Z**	**p[1]**	**Effect size[2] (Absolute r)**
H1: PSP	0-80	48.46	57.00	24.07	69.46	74.00	11.34	-4.101	<.001	Large (0.57)
H2: Time	sec.	1132.61	1085.00	354.64	574.05	626.10	214.06	-4.254	<.001	Large (0.59)
H3: PEOU	1-7	3.63	3.67	1.53	4.09	3.83	1.64	-1.196	.232	Small (0.16)

[1] Significant at p<.05 (Wilcoxon signed-rank test, 2-tailed).

[2] Effect size: small for $0.1 \leqslant |r| < 0.3$ and large for $|r| \geqslant 0.5$ (Cohen 1988).

Variables: ***PSP:*** Problem-solving performance; ***Time***: Time required; ***PEOU***: Perceived ease-of-use.

To further substantiate the support for H1, we performed follow-up tests for each of the six tasks. All test results showed significant effects (p=.001 for tasks 1 and 2, and p<.001 for tasks 3 through 6). Because of multiple comparisons we must apply a Bonferroni correction to control for inflated type I error rates.

Then, the tests must be significant at the .05/6=.008 level. Even under this consideration, all tests indicated significant differences for the conditions.

Because of our repeated measures design, learning and fatigue might have confounded the observed effects. Therefore, we additionally formed two groups of observations and compared their results. The first group contained all observations for the first experimental run (either table or ontology), while the second group those for the second experimental run (again, either table or ontology). Then, we ran Mann-Whitney U-tests. First, experimental run did not affect problem-solving performance (p=.980 for using the table and p=.268 for using the ontology). Second, the time required was lower in the second experimental run for both table (M=967.40, SD=247.95 vs. M=1297.81, SD=376.05, with p=.029) and ontology (M=458.06, SD=129.39 vs. M=690.03, SD=222.66, with p=.006), but no differential learning rate was observed. In summary, the results from our post-tests of confounding effects back up the findings from testing our hypotheses.

Discussion

We first discuss the implications of our study for practice and research and then its limitations.

Implications

Our research has important implications for practice. First, our evaluation provides evidence for the QoS aggregation ontology effectively supporting process designers in solving service replacement tasks. We observed a large size effect on problem-solving performance, with the mean accuracy increasing from about 61% when using the QoS aggregation table to about 87% when using the ontology. Although the participants worked with a limited set of service descriptions in a controlled and thus artificial environment, we designed tasks for a wide range of possible QoS parameters and observed the same effect for each task. The results are of importance to service selection tasks for heterogeneous QoS parameters because the ontology provides decision support for human process designers, which is complementary to prior design science research that seeks to automate the service replacement task for homogeneous QoS parameters.

Second, our evaluation provides evidence that using the ontology enables process designers to complete service replacement tasks more quickly. We observed a decrease from about 28 seconds per parameter when using the table to about 14 seconds per parameter. This finding is of importance for practice because such tasks at process runtime are time-critical, thus process designers must decide quickly whether a candidate service fits into the service slot. Making this decision would usually be aided by a service discovery component, which identifies only candidate services that meet the functional requirements. However, the mapping of QoS parameters onto each other is still a manual ad-hoc task and its performance largely depends on the process designer's ability to understand the semantics of heterogeneous service descriptions.

Third, our data suggests that the ontology can effectively be used by novice process designers (our participants were students and received limited training). We observed for both treatment conditions a medium level of perceived ease-of-use. Contrary to our expectation, ontology users did not experience higher level of ease-of-use. This might be due to the fact that our participants learned each problem-solving approach only through the one-page tutorial and received feedback on their solution only once (example task).

Fourth, our experiment provides an example of how to enrich the generic *Type* classes by subclasses to account for parameters with different semantics but same aggregation formulae. That is, using the ontology does not require any modification to the classes *Parameter*, *UpperBound*, *LowerBound*, *Composition Pattern*, and *Aggregation Formula* including all the restrictions on the *Type* classes. While we defined nine subclasses for our experiment (as shown in Appendix A2), many more such subclasses could be added. For instance, let us consider that a particular workflow requires a distinction of bandwidth to indicate whether video or audio data is being processed. This distinction can be implemented by adding two subclasses, e.g., *BandwidthVideo* and *BandwidthAudio*, to the *Type3* class. Therefore, both new classes inherit all the QoS aggregation knowledge from their superclass, and, eventually, they can be used by the process designer to map actual QoS parameters onto the ontology.

Our findings also have implications for future research. First, the usefulness of the QoS aggregation ontology for *service selection* tasks could be studied. In such a setting, process designers would need to select from multiple candidate services the best one by considering the QoS parameter values. Researchers could explore how participants use ontology, i.e., by (1) annotating all services and using a service selection component to determine the best candidate, or (2) deliberately selecting the best match and then annotating only that service. Thus, future research could explore different levels of decision support.

Second, while we have focused on using the ontology through annotating services, the ontology's extensibility offered by the generic *Type* classes could also be validated. It would be of interest to understand how process designers: (1) identify QoS parameters that have no corresponding subclasses in the ontology, and (2) add new subclasses to the generic types.

Third, the QoS model in the current ontology is restricted to representing upper and lower bounds for QoS parameters. Future research could extend the ontology by adding subconcepts to allow for a fine-grained representation of non-deterministic parameter, e.g., subconcepts for mean, standard deviation, and other distribution properties.

Limitations

We note some limitations of our research as well. First, our experiment used students as surrogates for novice process designers. While the students possessed the required knowledge of process modeling and service description to succeed in the experiment, they lacked the experience and domain knowledge that professionals might bring to bear in solving the tasks. We chose a rather homogeneous sample of students with *some* knowledge of the domain to not confound the posited effects with individual factors that we could not control. Therefore, the results of our experiment can only be generalized for novice process designers and not necessarily for experienced practitioners.

Second, our experiment was limited in the number of tasks and QoS parameters. While we created service descriptions specific to the experiment, we consulted prior research for common QoS parameters and retrieved synonyms from *WordNet*; hence, we followed a systematic approach to define parameter classes and service descriptions of high validity.

Third, the scope of our experiment was further limited by the five parameter types (*Type* classes) as defined in the ontology that we evaluated. In principle, a new parameter type could be added by defining a new subclass for the *Parameter* class and then amending the definitions of all aggregation formula classes with restrictions for the new parameter type. However, our analysis of the extant literature provides no indication for additional parameter types, thus we retained the five types with no changes.

Conclusion

Service replacement is a problem-solving task made difficult by the heterogeneity of service descriptions. Process designers must match corresponding service parameters and consider the many combinations of the service's role in the workflow and parameter types to determine the correct QoS aggregation formulae. This research demonstrates how designers can be assisted in that task by representing problem-solving knowledge in a domain ontology. We argue that adding semantic annotations from service parameters to classes defined in the aggregation ontology eases the problem-solving process compared to using an aggregation table. Our experimental results suggests that using the ontology improves the task performance and reduces the task completion time. Based on these results, our research provides empirical evidence for the QoS aggregation ontology being a useful solution to the important organizational problem of services computing.

Acknowledgements

The work of J. Leukel has been partially supported by the Federal Ministry of Education and Research, Germany, under grant 16KIS0083. The work of V. Sugumaran has been partly supported by a 2016 School of Business Administration Spring/Summer Research Fellowship from Oakland University.

Appendix: Experimental Materials

A1 Participant data

Demographics:

- Gender (male/female)
- Age (years)
- Program of study
- Undergraduate and graduate credits.

Self-reported modeling knowledge:

Please indicate the extent to which you agree or disagree with the following statements (7-point scale from "strongly disagree" to "strongly agree"):

- "Overall, I am very familiar with process models such as EPK/EPC or BPMN."
- "I feel very confident in *understanding* process models created with EPK/EPC or BPMN."
- "I feel very competent in *using* EPK/EPC or BPMN for process modeling."

A2 Task description

The following screenshot shows the interface for tasks that used the table.

You have to replace the faulty service by the candidate service. Please identify for each parameter of the faulty service a corresponding parameter of the candidate service, if existing, and fill in the correct aggregation formulae. The composition pattern for the current task is: **ORDISC**

Faulty Service

Parameter Name	Value
Accuracy	99
Service Outage Period	25
Storage Volume	80
Availability	99
Bandwidth	80
Data Encryption	128

Candidate Service

Parameter Name	Value
Downtime	28
Accessibility	96
Exactness	96
Encryption	96
Input/Output Rate	42
Capacity	42

Your Solution

Parameter Name	Upper Bound	Lower Bound
Exactness	max	product
---------	---------	---------
---------	---------	---------
---------	---------	---------
---------	---------	---------
---------	---------	---------

Dropdown options: ---------, N/A, Accessibility, Capacity, Downtime, Encryption, Exactness, Input/Output Rate

Aggregation formula table: (click image to enlarge)

Type	Bound	Sequence	Loop	XORXOR	ANDAND	ANDDISC	ORDISC	ORDISC
1	Upper	*sum*	*linear*	*max*	*sum*	*sum*	*sum*	*sum*
	Lower	*sum*	*linear*	*min*	*sum*	*sum*	*min*	*min*
2	Upper	*sum*	*linear*	*max*	*max*	*max*	*max*	*max*
	Lower	*sum*	*linear*	*min*	*max*	*min*	*min*	*min*
3	Upper	*min*	*identity*	*max*	*min*	*min*	*sum*	*sum*
	Lower	*min*	*identity*	*min*	*min*	*min*	*min*	*min*
4	Upper	*product*	*power*	*max*	*product*	*product*	*max*	*max*
	Lower	*product*	*power*	*min*	*product*	*product*	*product*	*product*
5	Upper	*min*	*identity*	*max*	*min*	*min*	*max*	*max*
	Lower	*min*	*identity*	*min*	*min*	*min*	*min*	*min*

Type 1: Parameters that are always summed up along all deterministic paths of the workflow (e.g., cost of service execution). For non-deterministic paths, the aggregation depends on whether the lower or upper bound is calculated.
Type 2: Parameters for which the critical path is determined by the maximal values in parallel executions (e.g., duration of execution time).
Type 3: Parameters that denote a capacity (e.g., throughput).
Type 4: Parameters that denote a probability (e.g., availability).
Type 5: Parameters for which the critical path is determined by the minimal values in parallel executions (e.g., key length of the service's encryption algorithm).

The following screenshot shows the interface for tasks that used the ontology.

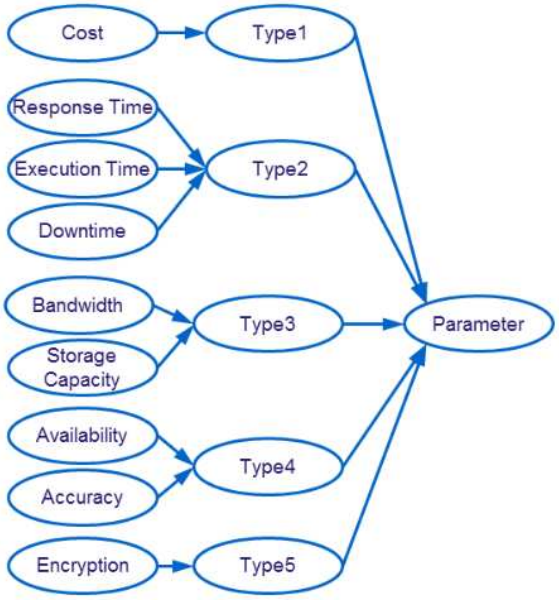
You have to replace the faulty service by the candidate service. Please select a class for each service parameter.

Faulty Service

Parameter Name	Value	Parameter Class
Accuracy	99	---------
Service Outage Period	25	---------
Availability	99	---------
Bandwidth	80	---------
Data Encryption	128	---------
Processing Duration	99	---------

Candidate Service

Parameter Name	Value	Parameter Class
Throughput	95	---------
Reliability	99	---------
Cycle Time	85	---------
Response Time	128	---------
Downtime	19	---------
Correctness	99	---------
Encryption	96	---------



Parameter type definitions:

Type 1: Parameters that are always summed up along all deterministic paths of the workflow (e.g., cost of service execution). For non-deterministic paths, the aggregation depends on whether the lower or upper bound is calculated.
Type 2: Parameters for which the critical path is determined by the maximal values in parallel executions (e.g., duration of execution time).
Type 3: Parameters that denote a capacity (e.g., throughput).
Type 4: Parameters that denote a probability (e.g., availability).
Type 5: Parameters for which the critical path is determined by the minimal values in parallel executions (e.g., key length of the service's encryption algorithm).

A3 Perceived ease-of-use

Please indicate the extent to which you agree or disagree with the following statements (7-point scale from "strongly disagree" to "strongly agree"):

- "Overall, I believe that the <<treatment condition>> was easy to use."
- "It was easy for me to remember how to perform tasks using the <<treatment condition>>."
- "Using the <<treatment condition>> was often frustrating."

References

Alrifai, M., Risse, T., and Nejdl, W. 2012. "A Hybrid Approach for Efficient Web Service Composition with End-to-End QoS Constraints," *ACM Transactions on the Web* (6:2), Article 7.

Ardagna, D., and Mirandola, R. 2010. "Per-Flow Optimal Service Selection for Web Services Based Processes," *Journal of Systems and Software* (83:8), pp. 1512-1523.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F. (eds.). 2010. *The Description Logic Handbook: Theory, Implementation, and Applications* (2nd ed.). New York: Cambridge University Press.

Basu, A., and Kumar, A. 2002. "Research Commentary: Workflow Management Issues in E-Business," *Information Systems Research* (13:1), pp. 1-14.

Beaty, H. W. 2006. "Units, Symbols, Constants, Definitions, and Conversion Factors," in D. G. Fink, and H. W. Beaty (eds.) *Standard Handbook for Electrical Engineers*, New York: McGraw-Hill Professional, pp. 1-57.

Becker, A., Widjaja, T., and Buxmann, P. 2011. "Value Potentials and Challenges of Service-Oriented Architectures – Results of an Empirical Survey from User and Vendor Perspective," *Business & Information Systems Engineering* (3:4), pp. 199-210.

Bhagat, J., Tanoh, F., Nzuobontane, E., Laurent, T., Orlowski, J., Roos, M., Wolstencroft, K., Aleksejevs, S., Stevens, R., Pettifer, S., Lopez, R., and Goble, C. A. 2010. "BioCatalogue: A Universal Catalogue of Web Services for the Life Sciences," *Nucleic Acids Research* (38:suppl 2), pp. W689-W694.

Blake, B., Cummings D. J., Bansal, A., and Bansal, S. K. 2012. "Workflow Composition of Service Level Agreements for Web Services," *Decision Support Systems* (53:1), pp. 234-244.

Canfora, G., Di Penta, M., Esposito, R., Perfetto, F., and Villani, M. L. 2008. "A Framework for QoS-Aware Binding and Re-Binding of Composite Web Services," *Journal of Systems and Software* (81:10), pp. 1754-1769.

Cardoso, J., Sheth, A. P., Miller, J. A., Arnold, J., and Kochut, K. 2004. "Quality of Service for Workflows and Web Service Processes," *Journal of Web Semantics* (1:3), pp. 281-308.

Chen, D. L., Schonger, M., and Wickens, C. 2016. "oTree – An Open-Source Platform for Laboratory, Online, and Field Experiments," *Journal of Behavioral and Experimental Finance* (9), pp. 88-97.

Cohen, J. 1988. *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.), Hillsdale, NJ: Lawrence Erlbaum Associates.

Dobson, G., Lock, R., and Sommerville, I. 2005. "QoSOnt: A QoS Ontology for Service-Centric Systems," in *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO '05)*, New York: IEEE Press, pp. 80-87.

Fakhfakh, N., Verjus, H., Pourraz, F., and Moreaux, P. 2013. "QoS Aggregation for Service Orchestrations Based on Workflow Pattern Rules and MCDM Method: Evaluation at Design Time and Runtime," *Service Oriented Computing and Applications* (7:1), pp. 15-31.

Faul, F., Erdfelder, E., Lang, A. G., and Buchner, A. 2007. "G* Power 3: A Flexible Statistical Power Analysis Program for the Social, Behavioral, and Biomedical Sciences," *Behavior Research Methods* (39:2), pp. 175-191.

Friedrich Schiller University Jena. 2015. *Jena Geography Dataset* (online at http://fusion.cs.uni-jena.de/fusion/activity/jena-geography-dataset retrieved September 4, 2016).

Gold, N., Mohan, A., Knight, C., and Munro, M. 2004. "Understanding Service-Oriented Software," *IEEE Software* (21:2), pp. 71-77.

Haq, I. ul, Huqqani, A., and Schikuta, E. 2011. "Hierarchical Aggregation of Service Level Agreements," *Data & Knowledge Engineering* (70:5), pp. 435-447.

Hevner, A. R., March, S. T., Park, J., and Ram, S. 2004. "Design Science in Information Systems Research," *MIS Quarterly* (28:1), pp. 75-105.

Horrocks, I., Patel-Schneider, P. F., and Van Harmelen, F. 2003. "From SHIQ and RDF to OWL: The Making of a Web Ontology Language," *Web Semantics: Science, Services and Agents on the World Wide Web* (1:1), pp. 7-26.

Huang, A. F., Lan, C.-W., and Yang, S. J. 2009. "An Optimal QoS-Based Web Service Selection Scheme," *Information Sciences* (179:19), pp. 3309-3322.

Hwang, S. Y., Wang, H., Tang, J., and Srivastava, J. 2007. "A Probabilistic Approach to Modeling and Estimating the QoS of Web-Services-Based Workflows," *Information Sciences* (177:23), pp. 5484-5503.

Iyer, B., Freedman, J., Gaynor, M., and Wyner, G. 2003. "Web Services: Enabling Dynamic Business Networks," *Communications of the Association for Information Systems* (11), Article 30.

Jaeger, M. C, Rojec-Goldmann, G., and Mühl, G. 2004. "QoS Aggregation for Web Service Composition Using Workflow Patterns," in *Proceedings of the 8th IEEE Enterprise Distributed Object Computing Conference (EDOC 2004),* New York: IEEE Press, pp. 149-159.

Karaenke, P., Leukel, J., and Sugumaran, V. 2013. "Ontology-Based QoS Aggregation for Composite Web Services," in *Proceedings of the 11th International Conference on Wirtschaftsinformatik (WI 2013),* Leipzig, Germany.

Khatri, V., Vessey, I., Ramesh, V., Clay, P., and Park, S.-J. 2006. "Understanding Conceptual Schemas: Exploring the Role of Application and IS Domain Knowledge," *Information Systems Research* (17:1), pp. 81-99.

Krishnan, M., Kumar, S., and Dakshinamoorthy, V. 2007. "SOA and Information Sharing in Supply Chain: 'How' Information is Shared Matters!," In *Proceedings of the 28th International Conference on Information Systems (ICEIS 2007)*, Paper 11.

Kritikos, K., Pernici, B., Plebani, P., Cappiello, C., Comuzzi, M., Benrernou, S., Brandic, I., Kertész, A., Parkin, M., and Carro, M. 2013. "A Survey on Service Quality Description," *ACM Computing Surveys* (46:1), Article 1.

Li, J., Ma, D., Mei, X., Sun, H., and Zheng, Z. 2011. "Adaptive QoS-Aware Service Process Reconfiguration," In *Proceedings of the 2011 IEEE International Conference on Services Computing (SCC 2011),* New York: IEEE Press, pp. 282-289.

Lin, K. J., Zhang, J., Zhai, Y., and Xu, B. 2010. "The Design and Implementation of Service Process Reconfiguration with End-to-End QoS Constraints in SOA," *Service Oriented Computing and Applications* (4:3), pp. 157-168.

Liu, A., Li, Q., Huang, L., and Xiao, M. 2010. "FACTS: A Framework for Fault-Tolerant Composition of Transactional Web Services," *IEEE Transactions on Services Computing* (3:1), pp. 46-59.

Liu, M., Wang, M., Shen, W., Luo, N., and Yan, J. 2012. "A Quality of Service (QoS)-Aware Execution Plan Selection Approach for a Service Composition Process," *Future Generation Computer Systems* (28:7), pp. 1080-1089.

Mendling, J., Reijers, H. A., and Recker, J. 2010. "Activity Labeling in Process Modeling: Empirical Insights and Recommendations," *Information Systems* (35:4), pp. 467-482.

Mendling, J., Strembeck, M., and Recker, J. 2012. "Factors of Process Model Comprehension — Findings from a Series of Experiments," *Decision Support Systems* (53:1), pp. 195-206

Moore, G. C., and Benbasat, I. 1991. "Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation," *Information Systems Research* (2:3), pp. 192-222.

Mukherjee, D., Jalote, P., and Nanda, M. G. 2008. "Determining QoS of WS-BPEL Compositions," in *Proceedings of the 6th International Conference on Service-Oriented Computing (ICSOC 2008),* A. Bouguetthaya, I. Krueger, and T. Margaria (eds.), Berlin: Springer, pp. 378-393.

Muñoz Frutos, H., Kotsiopoulos, I., Vaquero, L. M., and Merino, L. R. 2009. "Enhancing Service Selection by Semantic QoS," in *Proceedings of the 6th European Semantic Web Conference (ESWC 2009)*, Springer: Berlin, pp. 565-577.

O'Sullivan, J., Edmond, D., and ter Hofstede, A. H. M. 2002. "What's in a Service?," *Distributed and Parallel Databases* (12:2/3), pp. 117-133.

Oh, L.-B., Teo, H.-H., Leong, Y.-X., and Ravichandran, T. 2007. "Service-Oriented Architecture and Organizational Integration: An Empirical Study of IT-Enabled Sustained Competitive Advantage," In *Proceedings of the 28th International Conference on Information Systems (ICIS 2007)*, Paper 92.

Princeton University. 2016. *WordNet. A Lexical Database for English* (online at http://wordnet.princeton.edu; retrieved September 4, 2016).

Reijers, H. A., and Mendling, J. (2011). "A Study into the Factors that Influence the Understandability of Business Process Models," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* (41:3), 449-462.

Roa, H. N., Sadiq, S., and Indulska, M. 2014. "Ontology Usefulness in Human Tasks: Seeking Evidence," in *Proceedings of the 25th Australasian Conference on Information Systems (ACIS 2014)*, Auckland, New Zealand.

Smith, E., and Medin, D. 1981. *Categories and Concepts*, Cambridge, MA: Harvard University Press.

Stein, S., Payne, T. R., and Jennings, N. R. 2009. "Flexible Provisioning of Web Service Workflows," *ACM Transactions on Internet Technology* (9:1), pp. 1-45.

Sun, S. X., and Zhao, J. 2012. "A Decomposition-Based Approach for Service Composition with Global QoS Guarantees," *Information Sciences* (199), pp. 138-153.

Tran, V. X., Tsuji, H., and Masuda, R. 2009. "A New QoS Ontology and Its QoS-Based Ranking Algorithm for Web Services," *Simulation Modelling Practice and Theory* (17:8), pp. 1378-1398.

van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. 2003. "Workflow Patterns," *Distributed and Parallel Databases* (14:1), pp. 5-51.

W3C. 2004. *OWL Web Ontology Language Reference* (online at http://www.w3.org/TR/owl-ref; retrieved September 4, 2016).

Yu, T., Zhang, Y., and Lin, K.-J. 2007. "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Transaction on the Web* (1:1), Article 6.

Zeng, L., Benatallah, B., Ngu, A. H. H., Dumas, M., Kalagnanam, J., and Chang, H. 2004. "QoS-Aware Middleware for Web Services Composition," *IEEE Transactions on Software Engineering* (30:5), pp. 311-327.

Zhao, J. L., Tanniru, M., and Zhang, L.-J. 2007. "Services Computing as the Foundation of Enterprise Agility: Overview of Recent Advances and Introduction to the Special Issue," *Information Systems Frontiers* (9:1), pp. 1-8.

Zheng, H., Zhao, W., Yang, J., and Bouguettaya, A. 2013. "QoS Analysis for Web Service Compositions with Complex Structures," *IEEE Transactions on Services Computing* (6:3), pp. 373-386.