University of Huddersfield Repository

Baryannis, George, Kritikos, Kyriakos and Plexousakis, Dimitris

A specification-based QoS-aware design framework for service-based applications

Original Citation

Baryannis, George, Kritikos, Kyriakos and Plexousakis, Dimitris (2017) A specification-based QoS-aware design framework for service-based applications. Service Oriented Computing and Applications. ISSN 1863-2386

This version is available at http://eprints.hud.ac.uk/id/eprint/32104/

CrossMark

A specification-based QoS-aware design framework for service-based applications

George Baryannis[1] (ORCID) · **Kyriakos Kritikos[2]** · **Dimitris Plexousakis[2]**

Abstract Effective and accurate service discovery and composition rely on complete specifications of service behaviour, containing inputs and preconditions that are required before service execution, outputs, effects and ramifications of a successful execution and explanations for unsuccessful executions. The previously defined Web Service Specification Language (WSSL) relies on the fluent calculus formalism to produce such rich specifications for atomic and composite services. In this work, we propose further extensions that focus on the specification of QoS profiles, as well as partially observable service states. Additionally, a design framework for service-based applications is implemented based on WSSL, advancing state of the art by being the first service framework to simultaneously provide several desirable capabilities, such as supporting ramifications and partial observability, as well as non-determinism in composition schemas using heuristic encodings; providing explanations for unexpected behaviour; and QoS-awareness through goal-based techniques. These capabilities are illustrated through a comparative evaluation against prominent state-of-the-art approaches based on a typical SBA design scenario.

1 Introduction and motivation

The paradigm of *service-oriented computing (SOC)* is based on abstracting away from traditional software delivery models and considering software and related data as services available on demand [32], while promoting design principles such as reusability and composability. Such principles directly depend on the availability of rich service descriptions which cover both functional and non-functional aspects and which must be written in a formal, well-defined language in order to allow for automated discovery, verification and composition of the produced specifications [35]. Specifying service behaviour formally relies, among others, on expressing conditions that should hold before and after service execution, which gives rise to a family of problems, known in the AI literature as the frame, ramification and qualification problems [31]. These problems concern themselves with the representation of non-effects, knock-on or indirect effects and external or unforeseen preconditions, respectively.

To address these problems and achieve what can be termed as *representation completeness*, we previously proposed the Web Service Specification Language (WSSL) [6]. WSSL is independent of specific service design models and is founded on the fluent calculus [37], a formalism that offers integrated solutions to the frame, ramification and qualification problems. Additionally, we introduced an extension to WSSL [7] to enable the specification and generation of service compo-

✉ George Baryannis
g.bargiannis@hud.ac.uk

Kyriakos Kritikos
kritikos@ics.forth.gr

Dimitris Plexousakis
dp@ics.forth.gr

[1] Department of Informatics, Faculty of Computing and Engineering, University of Huddersfield, Queensgate, Huddersfield, UK

[2] Information Systems Laboratory, Institute of Computer Science, Foundation for Research and Technology - Hellas, N. Plastira 100, 70013 Heraklion, Crete, Greece
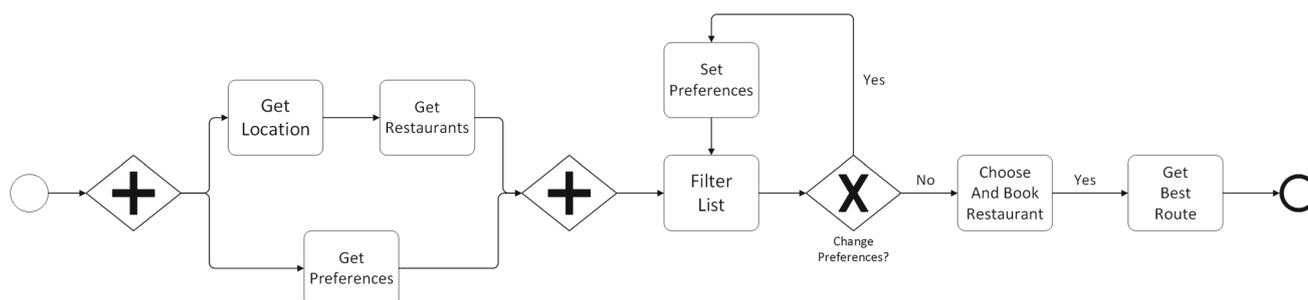
☷ Springer

Fig. 1 Composite process for the motivating scenario

sitions via planning based on logic programming in the fluent calculus, including support for non-deterministic patterns.

While the initial definition of WSSL allows for a complete representation of the behaviour of any service-based application (SBA), it is not yet capable of dealing with the real-world needs of designing SBAs based on an expressive specification language. To illustrate these needs, we present a smart mobility and tourism scenario, first defined within the CHOReVOLUTION project [29]. In this scenario, tourists require assistance in finding a place to eat, given their location and profile. A suitable SBA should search for matching nearby venues, allowing the user to modify preferences if the results are not satisfactory; the SBA should then book the chosen venue and compute the best route to it. Figure 1 illustrates the ideal composite process for this SBA. Based on the description in [29], the following requirements are extracted:

1. *Functional* (a) In terms of inputs, outputs and conditions, the following 4 should be supported: the designed SBA should output a feasible route to a nearby restaurant satisfying user's preferences, given the user's name as an input, under the condition that the user has activated GPS. (b) The SBA needs to combine functionality from existing services, ensuring that no conflicts arise between their preconditions and direct or indirect effects. (c) Complex and non-deterministic control constructs, such as parallel and conditional execution, should be exploited where possible. (d) The designed SBA should assume users may have partial/limited knowledge given that they are visitors in the area.
2. *Non-functional* The following 8 constraints need to be satisfied: filtering available restaurants and route calculation times should not exceed 3 and 5 s, respectively, while only free location services that achieve 99% availability should be used; overall throughput should be at least 100 requests per second, and all services included must be secured using the X.509 protocol and must support exception handling as well as the operation semantics of executing each request at least once.
3. *Other* (a) Executions of the designed SBA should be verifiable, providing explanations for abnormal cases

and considering unforeseen circumstances (e.g. system failure). (b) Alternative bindings for services should be considered in the designed SBA. (c) Services that violate functional or non-functional requirements should be discarded early. (d) If more than one SBA realise the scenario, a means of ranking them should be provided.

To satisfy requirements 1a-c and 3a, a *representationally complete* language that supports *correctness* verification, such as WSSL, is necessary since languages such as OWL-S [28] or WSML [38] are unable to: (1) model and reason about indirect effects, e.g. that booking a table in a restaurant indirectly leads to a request for computing the best route to reach it; (2) verify that the SBA executed correctly, or explaining why it did not, e.g. explain that failure is due to an error in the user's GPS or the restaurant booking system. If these requirements are not satisfied, the resulting SBAs would only capture part of the intended functionality and no troubleshooting would be possible in case of an unsuccessful execution. Requirement 1d can only be satisfied by a service composition approach that supports *partial observability*, capable of producing composite services even under incomplete knowledge, e.g. even without knowing whether the user's GPS is active. Failing to address this requirement leads to designed SBAs that are only applicable when there is complete knowledge of all states from start to finish, which cannot always be expected. Finally, to satisfy requirements 2 and 3b-d, the composition approach should also be *QoS-aware* and support ranking of produced SBAs.

While there have been numerous approaches that either possess one or only some of the highlighted features (more details in Sect. 2), to the best of our knowledge there is no SBA design framework that is capable of *simultaneously* supporting QoS-awareness, partial observability and correctness, not to mention relying on a representationally complete language. As argued by Lemos et al. [25], attempting to combine heterogeneous components relying on different, often semantically unrelated languages and notations leads to fragmentation of modelling, analysis and reasoning and breaks the maxim of sound design and engineering. Instead, we propose to build an integrated, end-to-end framework for SBA design that supports all the aforementioned features by rely-

ing on an extended version of WSSL which can achieve a unified representation of all service-related aspects.

Two WSSL extensions are introduced. The first involves expressing non-functional aspects through the specification of QoS profiles, additionally formalising correctness verification and defining aggregation formulas. The second extension deals with partial observability by allowing for the modelling of incomplete service states via constraints, as well as introducing the formalisation of *knowledge states*, i.e. states that may be possible according to our state knowledge. The proposed framework, named WSSL/SDF, makes full use of the extended version of WSSL and is accompanied by WSSL/TOOLS, a toolset to assist designers who have little to no knowledge of employing such a highly expressive language. WSSL/SDF combines the following features in an innovative and unified manner (while satisfying the scenario requirements shown in brackets):

– Composition allows for service and goal specifications that include preconditions, effects, ramifications and complex control constructs (reqs. 1a-c).
– The produced SBAs are automatically verified via proofs of correctness, also providing explanations for unexpected observed behaviour (req. 3a).
– QoS-awareness is achieved by pruning based on task-specific QoS goals and optimisation based on global QoS goals and aggregation formulas (req. 2).
– SBAs can be produced even when some states (e.g. the initial one) are partially observable (req. 1d).
– Functional discovery is realised via behavioural state-based matchmaking, ranking functionally equivalent SBAs based on a combination of domain-independent and problem-specific criteria. Produced SBAs can be dynamically linked (at run-time) to different service implementations, provided these conform to the respective WSSL specifications (reqs. 3b-d).

The remainder of this article is structured as follows. Section 2 offers a concise analysis of related work, while Sect. 3 summarises the formal definition of WSSL, as presented in [6,7]. Section 4 introduces WSSL extensions for the specification of QoS profiles and handling partial observability through knowledge modelling. WSSL/SDF and WSSL/TOOLS are defined and analysed in Sect. 5, while Sect. 6 evaluates how WSSL/SDF improves on state of the art. Section 7 concludes and points out future research directions.

2 Related work

The proposed framework spans a wide spectrum of topics within service science research, from description and design, to discovery, composition and verification. In this section, we focus only on research that is directly relevant to the stated contributions of this paper. Specifically, we focus on QoS-aware frameworks for services and SBAs, while also looking at the few research efforts that support some form of representation completeness.

2.1 QoS-aware service frameworks

One of the earliest comprehensive QoS-aware frameworks is PAWS [3], focusing on the adaptation and flexibility of service compositions modelled as business processes. Designers create a BPEL process which is then annotated with global and local constraints that usually refer to QoS aspects. For each task in the created process, a service retrieval module attempts to find services that have the required interface and do not violate any constraints. If no exact interface matches are found, a mediator is used to reconcile the interface discrepancies. PAWS also supports self-healing, allowing for faulty services to be substituted by other candidate services and at the same time enabling recovery actions to undo the results of the faulty services. As is common with earlier service composition frameworks, PAWS relies only on WSDL [10] interfaces, disregarding preconditions and postconditions.

Subsequent works successfully integrated conditions in service composition, with the most prominent examples being [24] and [4], both relying on OWL-S. [24] uses so-called causal links to determine whether functional or non-functional characteristics of services match. A causal link matrix (CLM+) is constructed, containing all possible matches and the composition approach relies on it to incrementally satisfy the service request, starting from the required outputs and postconditions and working backwards. Further work [12] expands on this approach, but drops support for non-functional aspects. [4] proposes a hierarchy-based dynamic composition approach that relies on planning knowledge organised as task hierarchies and decompositions. In contrast to HTN planning and similarly to our approach, a task at any level of granularity can be mapped to a concrete service. Pruning is performed on both task and plan levels based on QoS constraints to reduce search space and increase performance. Our approach improves on these techniques by adding a ranking phase so that, potentially, not all candidate plans go through the QoS-based selection process. Additionally, in contrast to our approach, neither [24] nor [4] consider plans more complex than trivial sequences.

A few QoS-aware frameworks manage to combine support for pre-/postconditions and complex control constructs. [39] proposes a composition schema generation process that focuses separately on data flow, functional and non-functional requirements. Both requests and produced compositions are modelled as statecharts, supporting parallel, conditional and iterative execution. [34] introduces VCL, a

composition language that supports both functional and non-functional constraints but, in contrast to WSSL, does not support either ramifications or qualifications. A structured composition that may include parallel and conditional control constructs is produced based on a dependency graph that models a VCL request. QoS-based selection is performed by either solving a constraint satisfaction problem or via integer programming techniques. Finally, Mabrouk et al. [26,27] rely on graph-based techniques to first produce a set of composition plans that satisfy functional and task-specific QoS constraints. Then, candidate services for each task are clustered according to the achieved quality levels and a selection algorithm aggregates QoS values for each plan and selects those that respect global QoS constraints.

A common characteristic of all the aforementioned approaches is that while they successfully achieve QoS-awareness, they do not address the equally important requirements of partial observability and correctness. At the same time, works that address these individually (indicatively see [8] for partial observability and [15] for correctness) do not combine them and are not QoS-aware. Indeed, these approaches cannot produce an SBA design if, for instance, the request does not fully specify the initial state; also, they do not offer any way of systematically verifying the behaviour of the produced compositions. Hence, they are unable to fully satisfy the requirements of the motivating scenario as analysed in Sect. 1. In contrast, WSSL/SDF can take into account partially specified information thanks to the extension described in Sect. 4.2 and can conduct verification of either the functionality of a plan (see Sect. 5.1) or the runtime behaviour of the produced SBA (see Section 5.4). To the extent of our knowledge, WSSL/SDF is the only QoS-aware service framework that can also support partial observability, correctness verification and representation completeness.

2.2 Representation completeness

None of the aforementioned frameworks takes into account any of the frame, ramification and qualification problems. The only other line of service research, to the extent of our knowledge, that attempts to achieve representation completeness is the work of Hoffmann et al. [16], which uses the possible models approach to address the representational facets of the frame and ramification problems. However, as stated by the authors, this solution is inadequate for problems that incorporate the notion of causality (such as the one in our motivating scenario); on the contrary, the solutions exploited by WSSL rely explicitly on causality and address both the representational and the inferential facet of the frame problem [36]; this allows any WSSL-based framework to be more effective in inferring state change. Furthermore, [16] does not address the qualification problem, and the composition approach is not QoS-aware and targets a restrictive

subset of services (which excludes services in the motivating scenario).

Few works have employed fluent calculus logic programming to compose services, with the most prominent being [18]. In contrast to our proposal, this work does not address the ramification and qualification problems, failing to capitalise on the benefits of their solutions. Moreover, no control constructs other than sequence and AND-Split/Join [17] are supported. Finally, inputs and outputs are represented using the fluent calculus constructs that model knowledge states. This can cause issues as it equates requiring an input and producing an output to fluents entailed by all possible states and creates conflicts when one attempts to model partially observable states (more in Sect. 4.2).

3 WSSL: web service specification language

This section provides a concise summary of WSSL, the language used as a foundation for the proposed SBA design framework.[1] WSSL uses the fluent calculus as its logical foundation; the basic fluent calculus notions follow, viewed from a service science perspective. A *fluent* is an atomic property which may change as a result of a service execution, represented by an *action*. A *state* is a fluent set that models a snapshot of the environment; macro $Holds(f, z)$ denotes that state z contains fluent f. A *situation* is a history of service executions, with function *State* mapping a situation to the state of the environment resulting after these executions. A first-order formula $\Delta(z)$ made of *Holds* expressions is called a *state formula*, if z is the only free state variable. A *situation formula* $\Delta(s)$ is defined equivalently.

The design of WSSL is driven by WSDL and Semantic Web services, but the language is capable of specifying any SBA behaviour that can be expressed in the form of conditions that should hold before and after service execution. Apart from the abstract syntax defined here, an XML syntax has also been defined in order to provide machine readability and facilitate exchange of WSSL documents on the Web.

A *WSSL specification* is defined as a 7-tuple $\mathcal{S} = \langle$**Service**, **Input**, **Output**, **Pre**, **Post**, **Causal**, **Default**$\rangle$. A description of these components follows. *Service* contains general information, e.g. service names and optional service grounding details. *Input* and *Output* represent inputs and outputs using two reserved unary fluent functions, $HasInput$ and $HasOutput$, denoting input availability and output production, as follows:

Definition 1 An *input formula* is a first-order formula $I(z)$ composed of *Holds* formulas on free state variable z, con-

---

sisting exclusively of $HasInput$ fluents. An *output formula* is defined equivalently.

Pre and *Post* represent preconditions and postconditions using axioms defined as follows:

Definition 2 An *action precondition axiom* for $A(x)$ is a formula $Poss(A(x), s) \equiv \Pi_A(x, s)$, meaning that action A is possible at situation s iff formula Π_A is true. A *state update axiom* is a formula $Poss(A(x), s) \rightarrow (\exists y)(\Delta(s) \wedge State(Do(A(x), s)) = State(s) + \theta^+ - \theta^-)$; its semantics is: provided that A is possible at situation s, execution results in a state produced by adding fluents that have been made true (*positive effects* θ^+) and subtracting falsified ones (*negative effects* θ^-), under optional additional conditions $\Delta(s)$.

For instance, inputs and outputs for the *Get Restaurants* task of the motivating scenario can be expressed as $HasInput(user\_location)$ and $HasOutput(rest\_list)$, respectively; its precondition is then $Located(username)$ and its postcondition is $Verified(rest\_list)$. Under the assumption that θ^+ and θ^- are disjoint (i.e. a service execution does not create and cancel the same effect), state update axioms are a provably correct solution to the frame problem (see Chapter 1 in [37]).

Causal contains a set of *causal relationships* which link effects to model ramifications (implicit, knock-on, or indirect effects). In contrast to the definition in [37], WSSL ramifications are not chained arbitrarily, since services are not expected to exhibit such behaviour.

Definition 3 A *causal relationship* is defined as a formula $(\forall)(\Gamma \rightarrow Causes(z, p, n, z', p', n', s)$ with the semantics that, under conditions expressed by formula Γ, positive and negative effects p and n that have occurred cause an update from state z to z', with positive and negative effects p' and n'. Inferring ramifications is expressed by the following macro: $Ramify(z, p, n, z', s) \stackrel{def}{=} (\exists p', n')(Causes(z - n + p, p, n, z', p', n', s)$.

For instance, to express the ramification of producing a routing request after a restaurant has been booked, we use the following: $Causes(z, Verified(booking), n, z + RouteRequest(booking), p + RouteRequest(booking), n, s)$. To address the qualification problem, unforeseen circumstances are modelled through the predicate $Acc(c, s)$ (accident c happened in situation s), e.g. define a $c = BookingError$ to represent failure to book a restaurant using the *Choose And Book* task of the motivating scenario. Finally, *Default* is a default theory formalising qualifications for service execution: *accidents* (i.e. unexpected situations) do not happen except if we cannot do otherwise. The simplest default theory only contains a universal default on the non-occurrence of all accidents.

To support composite service specification, fundamental control constructs (based on the patterns defined in [17]) are modelled using a set of function symbols, denoting sequence, conditional execution, AND-Split/Join, OR-Split/Join, XOR-Split/Join and iterative execution. Foundational axioms are defined for the newly introduced function symbols, encoding preconditions and postconditions for each control construct. For instance, $Poss(a_1 \cdot a_2, s) \rightarrow State(Do(a_1 \cdot a_2, s)) = State(Do(a_2, s)) + \theta_1^+ - \theta_1^- = State(s) + \theta_2^+ - \theta_2^- + \theta_1^+ - \theta_1^-$ states that AND-Split/Join results in a state where the effects of both services are applied.

4 WSSL extensions

This section proposes two extensions to the definition of WSSL, targeting specification of QoS profiles and handling partial observability via incomplete state and knowledge modelling.

4.1 Quality of service

By definition, any WSSL term can be associated with concepts defined in knowledge representation models, including ontology-based QoS models. For the purpose of QoS-awareness, we associate WSSL with quality models defined using OWL-Q [21], a semantic, rich and extensible meta-model for describing QoS attributes, metrics, requirements and capabilities.

WSSL specifications, as defined so far, are purely functional. Such specifications can be linked to one or more sets of QoS capabilities, collectively known as QoS profiles, representing the non-functional aspects exhibited by different implementations of the same functionality (e.g. multiple geolocation services with different QoS may exist for the *Get Location* task of the motivating scenario). Alternatively, different QoS profiles may refer to the same implementation, if we want to model *classes of service*, i.e. different QoS levels provided to consumers according to their needs and/or what they can afford. To support both cases, a WSSL specification needs to be linked to a QoS profile; to that end, we extend the definition in Sect. 3 to an 8-tuple, by including a *Quality* tuple which represents the aforementioned QoS profiles, defined as follows:

Definition 4 A QoS profile P is a non-empty, conjunctive set C of constraints of the form $<QoS\ term><comparison-operator><value>$, where a QoS term is an attribute or a metric, as defined in OWL-Q.

To represent decimal QoS values and comparison, we introduce a new sort named *DECIMAL* and a set of comparison operators: $\{<, \leq, \neq, \geq, >\}$ ($=$ is already part of WSSL expressions). For instance, a QoS profile for the *Get Best*

Route task of the motivating scenario would state that it achieves throughput of at least 50 requests/sec for a cost of 3 units, by including the following: $owlqmodel\#cost = 3 \wedge owlqmodel\#throughput \geq 50$, with *owlqmodel* referring to an OWL-Q QoS model. Local, task-based constraints would also need to be associated with the name of the task they correspond to. Note that we assume, for reasons of simplification, that all numeric values of QoS terms are decimals, with integers represented as decimals with a fractional part equal to zero. However, OWL-Q models may contain value type information attached to QoS terms which can be exploited by introducing additional constraints to represent the domain of values of these terms.

QoS attributes can be distinguished into measurable, using one or more QoS metrics, or unmeasurable ones, modelling static information that is qualitative in nature. While measurable attribute values are expressed using decimals, unmeasurable ones can also be represented as IRIs. For instance, robustness/flexibility, as defined in [33], can have the value set {*inflexible, flexible, very-flexible*}; these values can be modelled either by an IRI set, with one IRI per value, or by the set {0, 1, 2}, provided that a suitable mapping from possible attribute values to decimals exists.

WSSL QoS profiles can be extracted from SLA documents attached to concrete service implementations and expressed in languages such as WSLA [19] and WS-Agreement [1]. To achieve that, a baseline OWL-Q model can be used as a starting point, extending it with each new metric captured within the SLA document; all constraints contained in the document can then be straightforwardly converted to a WSSL QoS profile. To make sure that we do not add a metric that is equivalent to those already included, an alignment process has to be applied before adding each constraint, such as the one proposed in [22].

Given one or more QoS profiles, we need to identify (1) whether a profile is incorrect, leading to incorrect solutions, unsolvable problems or solving a different problem, and (2) whether one profile violates another, to determine whether a service implementation violates a QoS goal. The following definitions formalise these actions relying on constraint programming notions [13]:

Definition 5 Let P be a QoS profile, containing a set C of constraints and a constraint problem defined by C, a set V containing a single variable for each distinct term and a set D of domain values for each of these terms. Each domain value is derived by the respective term definition in the associated OWL-Q model. Let also S_P be the solution space (i.e. the set of all solutions) of the constraint problem associated with P. Then, P is *incorrect* iff S_P is an empty set.

Definition 6 Consider two QoS profiles P and G defined as in Definition 5, with solution spaces S_P and S_G, respectively.

Table 1 Correctness violations among constraints

Constraint 1	Constraint 2	Values Relation
$x = value_1$	$x = value_2$	$value_1 \neq value_2$
$x = value_1$	$x \neq value_2$	$value_1 = value_2$
$x = value_1$	$x < value_2$	$value_1 \geq value_2$
$x = value_1$	$x > value_2$	$value_1 \leq value_2$
$x = value_1$	$x \leq value_2$	$value_1 > value_2$
$x = value_1$	$x \geq value_2$	$value_1 < value_2$
$x < value_1$	$x > value_2$	$value_1 \leq value_2$
$x \leq value_1$	$x > value_2$	$value_1 \leq value_2$
$x \geq value_1$	$x < value_2$	$value_1 \geq value_2$
$x \leq value_1$	$x \geq value_2$	$value_1 < value_2$

P violates G if there is a solution in S_P that is not in S_G, expressed similarly to [11] as $S_P \cap S'_G \neq \emptyset$.

These definitions essentially translate to a pairwise check of constraints that refer to the same term, detecting violations when values are related as shown in Table 1, where terms are represented by variable x.[2]

4.1.1 QoS aggregation for composite SBAs

The WSSL QoS profiles discussed so far refer to atomic services. When such services are composed into an SBA, their individual profiles need to be considered in combination to infer a QoS profile for the complete SBA. This problem has been handled in the literature by aggregation functions depending on particular metrics and composition patterns. However, not all metrics supported by OWL-Q and not all composition patterns supported by WSSL are covered in existing work (e.g. [9,14,17,26,34]). To address this, we propose a set of generalised QoS aggregation functions, as shown in Table 2. x_a denotes the aggregated value, while x_i denotes the advertised (or measured) value for each atomic service, or the value of the worst-case bound, if a range of values is offered. In cases where alternative values are offered (some OR/XOR cases, reputation and throughput), the choice is left to the designer. Also, deterministic loops are treated as sequences of length equal to the maximum number of iterations, defined by the loop variant.

A brief analysis of the categories in Table 2 follows. Unmeasurable attributes are grouped based on their value type in the following three categories: (1) *Boolean*, containing properties that are either supported or not by a service (e.g. safety), for which aggregation involves determining whether the particular property is supported by all

---

[2] Definition 4 and Table 1 cover only single-valued terms. An extension to also include set operators is provided in Appendix B.1 (supplementary material).

Table 2 Categorisation and Aggregation of QoS terms

QoS term categories	Composition patterns		
	Sequence Det. loop	AND-Split/AND-Join	OR-Split/OR-Join XOR-Split/XOR-Join
Measurable			
Temporal	$x_a = \sum_{i=1}^{n} x_i$	$x_a = max\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$ or
Probabilistic	$x_a = \prod_{i=1}^{n} x_i$	$x_a = \prod_{i=1}^{n} x_i$	$x_a = max\{x_1, \ldots, x_n\}$
Throughput	$x_a = min\{x_1, \ldots, x_n\}$ or	$x_a = min\{x_1, \ldots, x_n\}$ or	$x_a = min\{x_1, \ldots, x_n\}$ or
	$x_a = max\{x_1, \ldots, x_n\}$	$x_a = max\{x_1, \ldots, x_n\}$	$x_a = max\{x_1, \ldots, x_n\}$
Cost	$x_a = \sum_{i=1}^{n} x_i$	$x_a = \sum_{i=1}^{n} x_i$	$x_a = \sum_{i=1}^{n} x_i$
Reputation	$x_a = avg\{x_1, \ldots, x_n\}$ or	$x_a = avg\{x_1, \ldots, x_n\}$ or	$x_a = avg\{x_1, \ldots, x_n\}$ or
	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$
Unmeasurable			
Boolean	$x_a = \begin{cases} false, & \exists i \cdot (x_i = false) \\ true, & otherwise \end{cases}$	$x_a = \begin{cases} false, & \exists i \cdot (x_i = false) \\ true, & otherwise \end{cases}$	$x_a = \begin{cases} false, & \exists i \cdot (x_i = false) \\ true, & otherwise \end{cases}$
Ordered set	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$	$x_a = min\{x_1, \ldots, x_n\}$
Unordered set	$x_a = \bigcap_{i=1}^{n} x_i$	$x_a = \bigcap_{i=1}^{n} x_i$	$x_a = \bigcap_{i=1}^{n} x_i$

services participating in the composition; (2) *Ordered Set*, e.g. robustness/flexibility [33], for which aggregation translates to finding the lowest-ordered one, which represents the lowest level shared by all participating services; and (3) *Unordered Set*, e.g. failure masking and operation semantics, as defined in [21], for which aggregation requires calculating the intersection of the value sets of all participating services.

While aggregation is pattern independent for unmeasurable attributes, this is not always the case for measurable attributes, which are categorised as follows:

– *Temporal* For attributes/metrics of temporal nature (e.g. execution time), the aggregate value represents the worst-case scenario, i.e. the sum for sequential, the maximum for AND-Split/Join and a min-max pair of values for OR and XOR, since the selected branches are unknown at design time.
– *Probabilistic* The aggregation of attributes such as availability and completeness involves applying the multiplication rule of probability (i.e. taking the product of all values) for sequences and AND patterns and again a min-max pair for OR and XOR.
– *Throughput* For attributes/metrics such as throughput and bandwidth, aggregation corresponds to finding the process bottleneck, represented by the minimum or maximum value, for positively or negatively monotonic metrics, respectively.
– *Cost* For the attribute of cost, aggregation is directly dependent on the cost model of each service provider; for simplicity, we assume monthly fees are charged, which leads to additive aggregation regardless of the composition pattern.

– *Reputation* For this attribute, aggregation is performed by calculating either the minimum value, or the mean value, if constraints concern the average reputation achieved by the SBA as a whole.

4.2 Partial observability

The second proposed extension involves partial observability with respect to WSSL states. There are three main causes of partial observability in service specification: (1) atomic services with non-deterministic outcomes, i.e. due to a state update axiom containing disjunction in the implication consequent; (2) composite services including conditional and non-deterministic iterative control constructs (like in Fig. 1); (3) partial knowledge of a state (initial or otherwise), e.g. in our motivating scenario, we do not know beforehand whether users will modify their preferences. Failing to recognise such cases leads to SBAs that partially ignore the expected behaviour of the SBA or composition approaches that cannot yield any results unless they are supplied with a complete specification of the initial state.

We propose two levels of handling partial observability. The lower level requires minimal modifications and is based on the generalised notion of an *incomplete state*, which is essentially a state defined using constraints. Three flavours of constraints are supported:

– *Negation* Constraints of the form $\neg Holds(f, z)$, suitable when we want to express that at a particular state, a property represented by f should not hold. For instance, to express that the *Filter List* task cannot be executed with an empty list as input, we use the constraint

$\neg Holds(Empty(rest\_list), z_{in})$, with z_{in} denoting the state before execution.

– *Universal quantification* A variant of the first case, formulated as $(\forall y)\neg Holds(f, z)$, that is applicable to fluents with multiple arguments to express that f does not hold for all values of argument y.

– *Disjunction* Constraints of the form $Holds(f_1, z) \vee \ldots \vee Holds(f_n, z)$, particularly useful to model states that may or may not involve a particular fluent. For instance, constraint $Holds(GPSActive, ?z\_in) \vee Holds(WiFiConnected, ?z\_in)$ states that the *Get Location* task may work using different sources.

State update axioms are slightly modified to support incomplete states with constraints: $Poss(A(x), s) \rightarrow (\exists y)(\Delta(s) \wedge (\exists z)(State(Do(A(x), s)) = \tau \circ z + \theta^+ - \theta^- \wedge \Psi))$. Essentially, we are certain that the fluents in τ hold in state z, the state following execution of A; for any other fluent to hold in z, it must satisfy constraints Ψ.

The higher level of handling partial observability involves modelling what it means to *know* that a fluent holds. A knowledge state is any state of the world that may be true according to what we know. If we have complete state knowledge, then there is only one possible state, the actual one; if we have no knowledge, all conceivable states are possible. The following definitions formalise knowledge states and expressions:

Definition 7 Formula $KState(s, z) \equiv \Phi(z)$, is a *knowledge state*, defining a possible state z in situation s, under a set of constraints expressed in state formula Φ.

Definition 8 A *knowledge expression* ϕ is known in a situation s based on the following: $Knows(\phi, s) \overset{def}{=} (\forall z)(KState(s, z) \rightarrow HOLDS(\phi, z))$ where ϕ may consist of fluents, stateless Poss predicates of the form $Poss(a)$ and atoms without state or situation terms, while $HOLDS(\phi, z)$ is obtained by replacing in ϕ all fluents with a Holds expression of the form $Holds(f, z)$ and adding state z to all Poss predicates.

Based on Definition 8, knowing f in situation s equates to $Knows(f, s) \overset{def}{=} (\forall z)(KState(s, z) \rightarrow Holds(f, z))$. In other words, f holds in all possible states at situation s. For example, to express that, as far as we know, location detection is activated at the start of the motivating scenario process, we use the knowledge expression $Knows(Holds(GelocActive(username), s_0))$, with s_0 representing the initial situation. Knowledge states are then used to redefine state update axioms to take possible states into account, albeit in a simplified way as compared to knowledge modelling in Chapter 5 of [37]. This is due to the fact that this modelling differentiates between physical and cognitive effects, which is relevant to autonomous robotic agents

but not SBAs. In the case of services, knowledge acquisition is neither a physical nor a cognitive effect, at least in the way defined for robotic agents. Thus, we propose to retain the initial way of modelling effects that lead from one state to the next: as an addition and/or subtraction of fluents.

Definition 9 A *knowledge update axiom* for action A is expressed as a formula $Poss(A(x), s) \rightarrow (\exists y)(\Delta(s) \wedge KState(Do(A(x), s), z') \equiv (\exists z)(KState(s, z) \wedge z' = z + \theta^+ - \theta^-))$. A *knowledge update axiom with ramifications* is defined accordingly based on the *Ramify* macro: $Poss(A(x), s) \rightarrow (\exists y)(\Delta(z) \wedge KState(Do(A(x), s), z') \equiv (\exists z)(KState(s, z) \wedge Ramify(z, \theta^+, \theta^-, z', Do(A(x), s))))$.

Essentially, handling partial observability amounts to replacing state update axioms with their knowledge update counterparts and employing knowledge states to express initial or subsequent states that are not completely defined. The SBA designer needs only define the constraints that represent this incomplete knowledge.

5 WSSL/SDF: SBA design framework

In this section, we present WSSL/SDF, a framework for designing SBAs based on WSSL specifications, using the extensions in Sect. 4 to support QoS-awareness and partial observability, while also exploiting WSSL to achieve representation completeness, automation and correctness verification. The envisioned SBA design approach using WSSL/SDF is shown in Fig. 2. A request is provided by an SBA designer in the form of a WSSL specification that describes the functional and non-functional aspects of the SBA. The request can be formed directly, if requesters are familiar with WSSL syntax, or via WSSL/TOOLS, as outlined in Sect. 5.5; in that case, they are assisted in the creation of WSSL documents, possibly relying on existing descriptions in other languages. The request is fed to the main framework components, whose objective is to facilitate the design process by offering a semi-automatic way of navigating through the numerous decisions within them. An analysis of these components follows.

5.1 Functional composition and verification

We adopt a divide-and-conquer approach, performing functional and non-functional composition separately, mainly to avoid the increased cost of backtracking in planning when considering all aspects at once (see also the relevant discussion in Chapter 6 of [37]). Separation also allows for the production of non-concretised processes, with tasks only associated with specifications; this is essential for supporting dynamicity, especially when we want to design SBAs that should achieve different, customer-specific QoS levels.
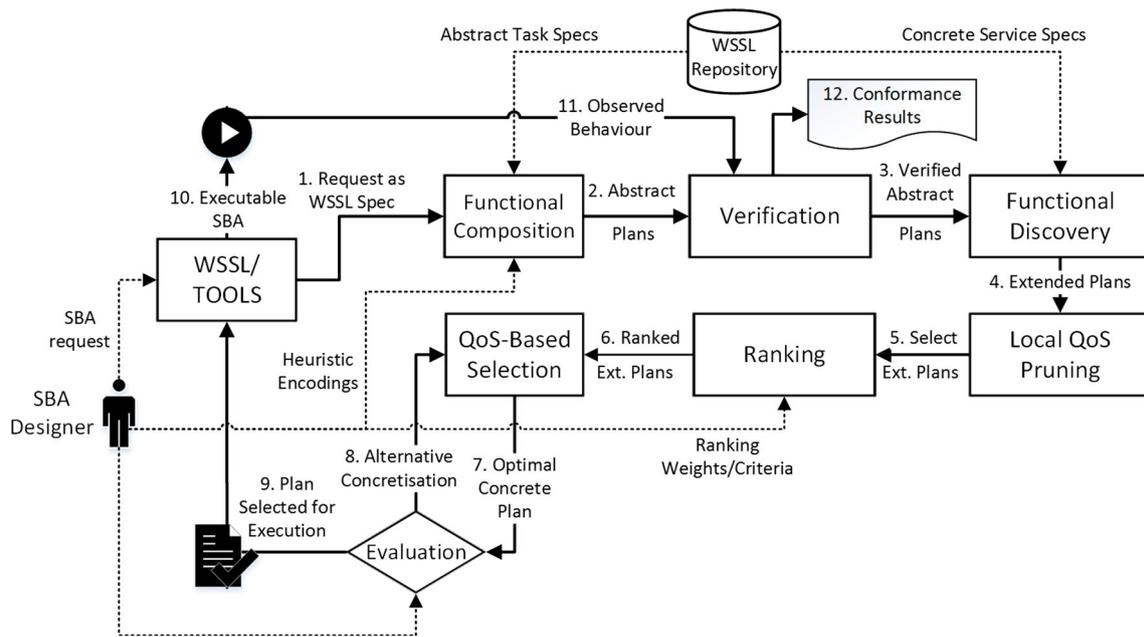
Fig. 2 Envisioned SBA design approach using WSSL/SDF

The first phase is realised using the implementation of WSSL in FLUX and the ECL^iPS^e constraint programming system [2]; it is a streamlined and updated version of the system presented in [7], where details on the formalisation of WSSL composition as a planning problem in FLUX can be found. The new version is again a customisation of the original FLUX kernel [37] and implements all (functional) WSSL features, applying the following: (1) simplifications to adapt to the service case and increase efficiency (e.g. simpler form of the *Ramify* macro, accidental updates modelled as disjuncts); (2) modifications, e.g. support for inputs and outputs in the definition of fluent addition; (3) extensions, e.g. inclusion of clauses that implement foundational axioms for control and data flow.

The partial observability extension of WSSL is implemented using the FLUX kernel as two sets of rules: (1) a set of constraint handling rules for negation, universal quantification and disjunction, and (2) rules that implement Definition 8 and variants for knowing that a fluent does not hold is a situation and for fluents with variables. These rules are applied by ECL^iPS^e only whenever state update axioms are expressed using constraints or when knowledge update axioms are used, for any service or request specification.

To reduce the overall planning time, we rely on two features: (1) a specially designed WSSL repository, where all specifications that share the same functional part are grouped under a single *abstract task*; thus, the planner considers each distinct functionality set only once; (2) heuristic encodings of the composition problem can be supplied, stating which control constructs (other than sequence) are to be considered at which parts of the process; WSSL/SDF

then only has to search for plans that conform to these encodings. Such encodings can be either derived by domain-specific knowledge (similar SBAs or past executions of WSSL/SDF) or explicitly defined by SBA designers, in case they have some particular expertise in the domain. For instance, an encoding that requires the parallel execution in Fig. 1 is expressed by the following Prolog rule (A1 stands for the sequence of *Get Location* and *Get Restaurants*): `plan(Z,[A|P],Z_PR):-A2=getprefs,A=xor(A1, A2),poss_xor(A1,A2,Z),state_update_xor(Z, A1,A2,Z_PR)` (Appendix A.2 gives a complete heuristic encoding for the motivating scenario).

After functional composition, an optional verification phase follows, checking conformance of the plans to certain properties, such as composability, ensuring there are no conflicts within a composition plan or liveness and safety properties, ensuring that a plan realises the requested behaviour. Thanks to WSSL's solution to the qualification problem, WSSL/SDF is able to indicate the task or tasks as well as the corresponding clauses in the heuristic encoding that caused the failure, assisting the designer in resolving such situations.

5.2 Specification-based functional discovery

Composition results in a set of *abstract plans*, each defined formally as a non-empty sequence $\alpha_1, \ldots, \alpha_n$, with each α corresponding to either an atomic abstract task T, or a composite one, formed by combining one or more abstract tasks using one or more control constructs. For each task in these plans, functional discovery is performed, yielding a set of

extended plans, each formally defined as an abstract plan where each contained atomic abstract task T is linked with a sequence $S_1, \ldots, S_n$ of concrete service implementations. Matchmaking is formalised as follows:

Definition 10 Given two WSSL specifications T (abstract task) and S (concrete service), S matches T iff the following hold:

- *Action precondition axioms* If $Poss(T(x), z) \equiv \Pi_T(z)$ and $Poss(S(x), z) \equiv \Pi_S(z)$, then $\Pi_S(z) \Rightarrow \Pi_T(z)$ must hold. With $\Pi_T(z) \equiv Holds(f_{T1}, z) \wedge ... \wedge Holds(f_{Tn}, z)$ and $\Pi_S(z) \equiv Holds(f_{S1}, z) \wedge ... \wedge Holds(f_{Sn}, z)$, then $\Pi_S(z) = \Pi_T(z) \equiv \{f_{Ti}, ..., f_{Tn}\} = \{f_{Si}, ..., f_{Sn}\}$.
- *State update axioms* Given $Poss(T(x), s) \Rightarrow (\exists y_T) (\Delta_T(s) \wedge State(Do(T(x), s)) = State(s) + \theta_T^+ - \theta_T^-)$ and a similar axiom for S, then $\Delta_S(s) \Rightarrow \Delta_T(s)$ and $State(Do(T(x), s)) = State(Do(S(x), s)) \equiv (\theta_T^+ = \theta_S^+) \wedge (\theta_T^- = \theta_S^-)$ must hold.
- *State update with ramifications* If $Poss(T(x), s) \Rightarrow (\exists y_T) \Delta_T(s) \wedge Ramify(z, \theta_T^+, \theta_T^-, z')$ and $Poss(S(x), s) \Rightarrow (\exists y_S) \Delta_S(s) \wedge Ramify(z, \theta_S^+, \theta_S^-, z')$ then $\Delta_S(s) \Rightarrow \Delta_T(s)$ and $(\theta_T^+ = \theta_S^+) \wedge (\theta_T^- = \theta_S^-)$ must hold, in addition to the following relation between causal relationships:
- *Causal relationships* If CR_T and CR_S are the sets of causal relationships contained in T and S, respectively, then $CR_T \subseteq CR_S$ must hold.

Definition 10 requires that both specifications refer to the same inputs and preconditions and lead to the same state after execution, based on exact matches between fluents.[3] Functional matchmaking is realised using WSSL's FLUX kernels. For example, FLUX clause `poss(T,Z),Z=[hasinput(usr-name), gpsactive(usrname)]` realises action precondition matching for candidates T of *Get Location*. Relying on specifications raises functional discovery to a higher level, disregarding implementation details. This is realisable only if the same specification language is used for all services, while specifications must refer to the same conceptual models; otherwise, alignment and transformation phases must precede discovery, based on translators among specification languages and adapters that modify a specification to refer to a different conceptual model.

5.3 Local QoS pruning and ranking

The number and size of extended plans produced in the discovery phase depend primarily on repository size, functional goal complexity and how elaborate heuristic encodings are. Pruning and ranking aim at decreasing the size and complexity of the extended plan set without compromising optimality.

---

[3] Subsumption, plug-in and other matches can also be supported, provided that such relations are expressed between fluents representing service IOPEs, e.g. stating that f_{Si} subsumes f_{Ti}.

Pruning removes from extended plans any concrete services that do not satisfy a local QoS goal (expressed within the initial SBA request), using Definition 6, with P representing the QoS profile of the concrete service and G the local goal. Solution spaces are compared according to Table 1 and relying on the unary technique introduced in [23]. This technique uses an ordered set of values for each metric in a constraint in G, based on the values offered in the QoS profiles. Using each of these sets, implementations that offer a greater value (for negatively monotonic metrics such as cost) or a lower one (for positively monotonic metrics such as availability) value than the one in G are pruned.

An extended plan is discarded if all implementations for a single task are pruned. The plans that survive the pruning process are then ranked to determine the order in which they are examined in the final phase. We opt to rank shorter plans higher, since they satisfy the SBA request in less steps; hence, two ranking criteria are maximum execution path length and total number of tasks. For both criteria, the calculated score is the reciprocal of the actual number (length or number of tasks), leading to maximum scores equal to 1 (for trivial, single-task plans). In addition to these criteria, there may be other domain/problem-dependent ones, e.g. in the motivating scenario, plans where the booking process is realised in a single task may be more preferable, so that sensitive payment-related information is not accessed by more than one service.

Problem-dependent criteria are expected to be defined by designers with experience on the particular domain or problem, in the form of specific tasks or sub-processes which the user demands (or prohibits) to be part of the plan. In the motivating scenario, for instance, plans that allow the user to change their preferences may be ranked higher. To produce a total rank score out of the individual scores of the ranking criteria, the widely used simple additive weighting method is employed. Weights are attributed uniformly by default, but other settings may be applicable depending on each problem; for instance, plan complexity may be less important compared to problem-specific ranking criteria, leading to higher ranking weights for the latter.

Both pruning and ranking are realised by exhaustively traversing extended plans, either to check for local QoS goal violations or to evaluate ranking criteria. If an extended plan comprises n abstract tasks, the complexity of the pruning and ranking processes is $\mathcal{O}(n)$.

5.4 QoS-based selection

Following pruning and ranking, global QoS-based selection is performed for the remaining extended plans, based on the QoS profiles included in the specifications of the services within these plans, until a combination is found that satisfies all global QoS goals. This combination is essentially an SBA

design that satisfies all functional and non-functional goals of the initial request. Global goal matching requires that the target value of each global goal is compared to an aggregation of the values achieved by the individual services within the plan; for instance, in the motivating scenario the SBA cost must be kept under a specific threshold, so the total cost of all services in the plan must be aggregated to determine whether it exceeds the threshold. We use the aggregation functions in Table 2, parsing plans from start to end in order to calculate values for all attributes related to global QoS goals.[4] QoS-based selection is then performed using existing algorithms in the literature. The current implementation of WSSL/SDF uses the algorithms defined in [20], assuming no knowledge of execution path probabilities within the plans.[5]

If performing QoS-based selection on the top-ranked plan does not yield a solution, the next-ranked plan undergoes the same process. In the best-case scenario, only the top-ranked extended plan will need to be examined, reducing overall execution time. Designers then evaluate the resulting concrete SBA, which can be transformed to BPMN or BPEL using WSSL/TOOLS, and executed. The observed run-time behaviour of the SBA can also be subjected to verification, to check whether it conforms to the initial request. In case conformance fails with regard to functional aspects, possible explanations can be derived (based on the qualification problem solution described in Sect. 3), in order to perform troubleshooting actions.

5.5 WSSL/TOOLS: modelling toolset

The primary aim of WSSL is to spread the idea of representation completeness, creating specifications that can effectively answer how a particular service affects the state of the world under any possible circumstance. However, such a highly expressive formal language requires significant modelling effort on the part of service designers, as well as a high level of expertise in service behavioural aspects, which, in turn, may hinder the usability of WSSL/SDF. In order to mitigate these effects, we propose to include a modelling toolset, called WSSL/TOOLS, within the framework. We have implemented a first prototype version[6] which supports the following features:

- *Import from WSDL and OWL-S* convert existing WSDL or OWL-S documents to equivalent WSSL ones, based on the grounding mechanism in [5].
- *Export to XML, FLUX or WSDL* WSSL/TOOLS can export a WSSL specification as a WSSL/XML document or as a FLUX program; it can also generate an equivalent WSDL file.
- *Validation support* detect conflicting information in a specification (e.g. preconditions that contradict each other) and assist in resolving such conflicts.

Using this version, designers can reuse existing service specifications, without being familiar with the WSSL syntax. However, to use WSSL features that are not included in languages such as WSDL and OWL-S, they still require knowledge of WSSL ramifications, default theories for WSSL qualifications and any other underlying fluent calculus notions. To avoid this and strengthen the capabilities of the toolset, we propose the addition of the following features:

- *Partially observable states* guide the user in the definition of incomplete or knowledge states, by assisting in writing constraints and state formulas that conform to WSSL.
- *Completion support* guide the user in completing a specification by adding any WSSL element, possibly with support of specification libraries.
- *Non-functional aspects* assist in creating QoS profiles, using suitable terms by imported OWL-Q models or SLA specifications, and in their correctness evaluation (based on Definition 5). Convert WSSL QoS profiles (e.g. for a produced composite SBA) to SLA specifications.
- *Export to BPMN and BPEL* provide assistance in viewing, editing and executing WSSL plans as BPEL or BPMN models.

With this expanded set of features, WSSL/TOOLS essentially masks formal language specifics that service designers are most likely unfamiliar with. Thus, it can potentially increase the usability of WSSL/SDF, since by relying on the aforementioned features, the required skills of a service designer in order to use the framework only amount to knowledge of basic service behavioural aspects (IOPEs and non-functional properties) and basic knowledge of predicate logic.

6 Evaluation

In this section, we evaluate WSSL/SDF by directly comparing it with the most prominent related efforts, specifically [34] and [26]. The rationale behind this choice is twofold: (1) these works are the ones simultaneously sat-

---

[4] The aggregation algorithm is available in Appendix B.2.

[5] If such knowledge is derived through an analysis on possible execution paths, then the algorithms in [30] are more suitable, since they include an aggregation process that relies on execution path probabilities.

[6] The prototype version of WSSL/TOOLS is available in http://www.csd.uoc.gr/~gmparg/wssl-sdf.

Table 3 WSSL Specifications for select tasks of the motivating scenario

Service	Inputs	Outputs	Qualifications
Get location	username	user_location	GeolocError
Choose and book	filtered_list	booking	BookingError
Get best route	booking, user_location	route	RoutingError

Service	Preconditions	Postconditions	Ramifications
Get location	GeolocActive(username)	Located(username)	–
Choose and book	Verified(filtered_list), ¬UpdRequest(username)	Verified(booking)	RouteRequest(booking)
Get best route	RouteRequest(booking)	Verified(route)	¬RouteRequest(booking)

Table 4 Optimality evaluation (motivating scenario)

Framework	Func.	NonFunc.	Other	Time (ms)
Mabrouk et al. [26]	5/7	5/8	3/4	322.93
Rosenberg et al. [34]	5/7	6/8	3/4	1480
WSSL/SDF	7/7	8/8	4/4	293.2

isfying the maximum amount of requirements, as analysed in Sect. 2; (2) they are the only ones, to the best of our knowledge, that rely on a unified representation language for functional and non-functional aspects. Due to these reasons, they can be considered the most prominent and relevant to this work, allowing us to perform a uniform and fair comparison. All experiments were performed on a Windows® 8 system with an Intel® Core™ i7-740QM processor at 1.73GHz, with 6 GB RAM. Calculated values are an average of 20 runs. We assume that we are tasked to design an SBA for the scenario stated in Sect. 1 and we want to investigate how WSSL/SDF improves on state of the art in terms of satisfying all requirements (functional, non-functional and other) declared there.

We also assume that available services are organised in a WSSL repository as defined in Sect. 5.1; a subset of the contained specifications is shown in Table 3 (the complete table is in Appendix A.1), expressed in a language-independent form so that they can be adapted to VCL and EASY-L, the languages used by Rosenberg et al. [34] and Mabrouk et al. [26], respectively. For each task, the repository contains 10 candidate services. The results are shown in Table 4 and are explained below. Since different experiment setups were used in [34] and [26] and their implementations are not accessible to rerun the experiments, we normalised reported results according to benchmark comparisons of the CPUs used.[7]

In terms of functional requirements, [34] and [26] are unable to recognise ramifications and cannot work with partially observable initial states; this means that they cannot link *Choose and Book* with *Get Best Route* through the

former's ramification and they cannot know if *Set Preferences* is to be included in the plan, unless the truth value of *UpdRequest(username)* is known beforehand. In terms of non-functional requirements, aggregation formulas in [26] do not recognise any unmeasurable attributes, while those in [34] do not recognise operation semantics and exception handling. Also, while all three approaches support alternative bindings and local QoS filtering, with [26] additionally supporting results ranking, only WSSL/SDF can provide verification support, including identification of causes in case of failure thanks to the qualification problem solution. For instance, if the service implementing the *Choose and Book* task fails to produce the expected results, the related cause *BookingError* is produced and picked up when the verification process in WSSL/SDF queries the current state after the execution of the SBA.

In terms of execution time, achieving optimality in the relatively simple case of the motivating scenario does not lead to decreased performance, since results are roughly equivalent to [26]. Note that [34] exhibits significantly higher execution times (attributed to their costly feature resolution phase), even if we take into account the pre-normalised value (627 ms, run on a powerful testbed with doubled CPU power compared to ours). Combined with the fact that the provided experiments in [34] show linear increases to execution time for more complex setups, this indicates that their approach will always achieve higher execution times. To that end, we exclude [34] from the next experiment, a decision validated by the results that follow.

The second experiment aims to evaluate the scalability of WSSL/SDF in direct comparison to the results in [26] to determine how performance is affected for problems more complex than the motivating scenario. The experiment settings were chosen based on the experiments in [26] and are as follows: specification size was fixed to 5 IOPEs and non-functional goals to 5 local and 5 global QoS constraints; for each task there are 10 candidate implementations and compositions are sequences that range from 10 to 50 tasks. We also assume the median case of one discovery run per 2 tasks in the sequence (as opposed to the best and worst cases of all

---

[7] Appendix C contains a detailed account of the normalisation process as well as additional performance experiments.
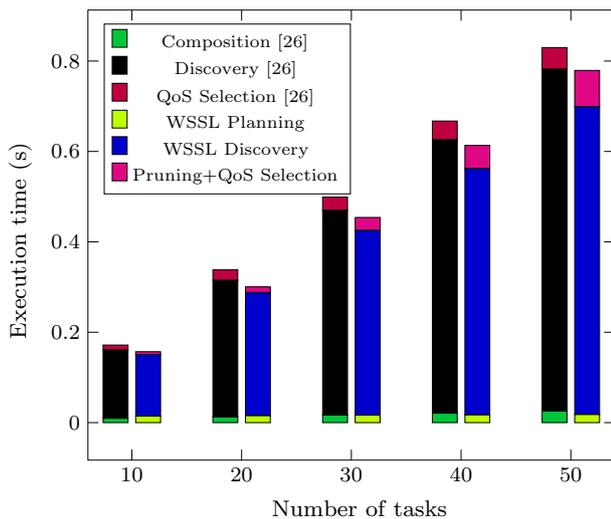
Fig. 3 Comparative scalability evaluation

tasks being equivalent or completely distinct, respectively). The results are shown in Fig. 3. Total execution time is again comparable to [26], slightly decreased by 8% on average, scaling reasonably with the number of tasks. Since planning and QoS selection times are roughly similar, the decrease is attributed mostly to the functional discovery phase. This may be due to the way matchmaking is implemented: in this particular setup, ECL^iPS^e Prolog queries seem to run slightly faster than the tableaux-based ones employed by Mabrouk et al. [26]. Based on the results of the evaluation, we can conclude that WSSL/SDF can achieve optimality in service design without compromising in terms of performance scalability.

7 Conclusions and future work

This article aims at illustrating the benefits of employing rich specifications taking into account representation problems, when describing and composing services during SBA design. The proposed end-to-end design framework is an example of how such specifications can help achieve a combination of desired capabilities, including dynamicity, QoS-awareness, non-determinism and partial observability. WSSL and WSSL/SDF can have a substantial impact to SOA stakeholders. Service providers can advertise their products more effectively using WSSL specifications, increasing their trustworthiness; service consumers are then more likely to achieve their goals. SBA designers can employ WSSL/SDF to reduce service design effort, at the cost of an increased effort in creating service specifications; WSSL/TOOLS offers a means towards managing this cost. The conducted evaluation shows that WSSL/SDF improves on state of the art by accurately capturing all requirements of a complex SBA design scenario.

Future research will follow several interesting directions. Primarily, a user study on WSSL/SDF needs to be conducted to evaluate modelling effort and usability. Concerning the capabilities of WSSL/SDF, we plan to implement the additional features of WSSL/TOOLS outlined at the end of Sect. 5.5 and also explore the following extensions: (1) alignment and normalisation when different ontologies are employed; (2) adaptation or mediation between different I/O concepts in order to achieve data integration; (3) discovery enhancement via ontology-based reasoning to discover relations between concepts; (4) support for n-ary constraints, as well as soft conditions and constraints in order to address over-constrained functional requirements.

References

1. Andrieux A, Czajkowski K, Dan A, Keahey K, et al (2011) Web services agreement specification (WS-Agreement). https://www.ogf.org/documents/GFD.192.pdf
2. Apt KR, Wallace M (2007) Constraint logic programming using ECLiPSe. Cambridge University Press, Cambridge
3. Ardagna D, Comuzzi M, Mussi E, Pernici B, Plebani P (2007) PAWS: a framework for executing adaptive web-service processes. IEEE Softw 24(6):39–46
4. Barakat L, Miles S, Poernomo I, Luck M (2011) Efficient multi-granularity service composition. In: 2011 IEEE international conference on web services ICWS '11. IEEE Computer Society, Washington, pp 227–234
5. Baryannis G (2014) A novel specification and composition language for services. Ph.D. dissertation, University of Crete, Greece, http://elocus.lib.uoc.gr/dlib/f/6/e/metadata-dlib-1415268652-781614-13324.tkl?lang=en
6. Baryannis G, Plexousakis D (2013) WSSL: a fluent calculus-based language for web service specifications. In: Salinesi C, Norrie MC, Pastor Ó (eds) Advanced information systems engineering: proceedings of the 25th international conference, CAiSE 2013, LNCS, vol 7908. Springer, Berlin, pp 256–271
7. Baryannis G, Plexousakis D (2014) Fluent calculus-based semantic web service composition and verification using WSSL. In: Lomuscio A et al (eds) Service-oriented computing ICSOC 2013 Workshops, LNCS, vol 8377. Springer International Publishing, Switzerland, pp 256–270
8. Bertoli P, Pistore M, Traverso P (2010) Automated composition of web services by planning in asynchronous domains. Artif Intell 174(3–4):316–361
9. Cardoso J, Sheth AP, Miller JA, Arnold J, Kochut K (2004) Quality of service for workflows and web service processes. J Web Semant 1(3):281–308
10. Chinnici R, Moreau JJ, Ryman A, Weerawarana S (2007) Web services description language (WSDL) Version 2.0 Part 1: core language. World Wide Web Consortium, Recommendation, http://www.w3.org/TR/wsdl20

11. Cortés AR, Martín-Díaz O, Toro AD, Toro M (2005) Improving the automatic procurement of web services using constraint programming. Int J Cooperative Inf Syst 14(4):439–468

12. Eshuis R, Lcu F, Mehandjiev N (2016) Flexible construction of executable service compositions from reusable semantic knowledge. ACM TWEB 10(1):5:1–5:27

13. Freuder EC, Mackworth AK (2006) Constraint satisfaction: an emerging paradigm. In: Rossi F, van Beek P, Walsh T (eds) Handbook of constraint programming, foundations of artificial intelligence, vol 2. Elsevier, New York, pp 13–27

14. Gao F, Curry E, Ali MI, Bhiri S, Mileo A (2014) QoS-aware complex event service composition and optimization using genetic algorithms. In: Franch X, Ghose AK, Lewis GA, Bhiri S (eds) Service-oriented computing ICSOC 2014, LNCS, vol 8831. Springer, Berlin, pp 386–393

15. Hilia M, Chibani A, Djouani K, Amirat Y (2012) Semantic service composition framework for multidomain ubiquitous computing applications. In: Liu C, Ludwig H, Toumani F, Yu Q (eds) Service-oriented computing ICSOC 2012, LNCS, vol 7636, Springer, Berlin, pp 450–467

16. Hoffmann J, Bertoli P, Helmert M, Pistore M (2009) Message-based web service composition, integrity constraints, and planning under uncertainty: a new connection. J Artif Intell Res (JAIR) 35:49–117

17. Jaeger MC, Rojec-Goldmann G, Mühl G (2004) QoS aggregation for web service composition using workflow patterns. In: Proceedings of the 8th IEEE international enterprise distributed object computing conference EDOC '04. IEEE Computer Society, Washington, pp 149–159

18. Karpagam G, Bhuvaneswari A (2011) AI planning-based semantic web service composition. Int J Innov Comput Appl 3(3):126–135. doi:10.1504/IJICA.2011.041913

19. Keller A, Ludwig H (2003) The WSLA framework: specifying and monitoring service level agreements for web services. J Netw Syst Manag 11(1):57–81

20. Kritikos K, Plexousakis D (2009a) Mixed-integer programming for QoS-based web service matchmaking. IEEE Trans Serv Comput 2(2):122–139

21. Kritikos K, Plexousakis D (2009b) Requirements for QoS-based web service description and discovery. IEEE Trans Serv Comput 2(4):320–337

22. Kritikos K, Plexousakis D (2012) Towards aligning and matchmaking QoS-based web service specifications. In: Reiff-Marganiec S, Tilly M (eds) Handbook of research on service-oriented systems and non-functional properties, chap 10. IGI Global, Hershey, pp 216–257

23. Kritikos K, Plexousakis D (2014) Novel optimal and scalable non-functional service matchmaking techniques. IEEE Trans Serv Comput 7(4):614–627

24. Lécué F, Silva E, Pires Ferreira L (2008) A framework for dynamic web services composition. In: Gschwind T, Pautasso C (eds) Emerging web services technology, vol II. Whitestein series in software agent technologies and autonomic computing. Birkhäuser Basel, Switzerland, pp 59–75

25. Lemos AL, Daniel F, Benatallah B (2016) Web service composition: a survey of techniques and tools. ACM Comput Surv 48(3):33

26. Mabrouk NB, Beauche S, Kuznetsova E, Georgantas N, Issarny V (2009) QoS-aware service composition in dynamic service oriented environments. In: Bacon J, Cooper BF (eds) Middleware 2009: proceedings of the 10th international middleware conference, LNCS, vol 5896, Springer, pp 123–142

27. Mabrouk NB, Georgantas N, Issarny V (2015) Set-based bi-level optimisation for QoS-aware service composition in ubiquitous environments. In: Zhu H, Miller JA (eds) IEEE international conference on web services ICWS' 15, IEEE Computer Society, pp 25–32

28. Martin D, Burstein M, et al (2004) OWL-S: semantic markup for web services. http://www.ai.sri.com/daml/services/owl-s/1.2

29. Masetti M, Naselli A, Keller S (2015) Smart mobility and tourism scenario definition and requirements. Tech. rep., CHOReVOLUTION H2020 ICT9 Project, http://www.chorevolution.eu

30. Mello Ferreira A, Kritikos K, Pernici B (2009) Energy-aware design of service-based applications. In: Baresi L, Chi CH, Suzuki J (eds) Service-oriented computing: proceedings of the 7th international joint conference, ICSOC-ServiceWave 2009, LNCS, vol 5900. Springer, Berlin

31. Miller R (2006) Three problems in logic-based knowledge representation. ASLIB Proc New Inf Perspect 58(1/2):140–151

32. Papazoglou MP, Traverso P, Dustdar S, Leymann F (2007) Service-oriented computing: state of the art and research challenges. Computer 40(11):38–45

33. Ran S (2003) A model for web services discovery with QoS. ACM Sigecom Exch 4(1):1–10

34. Rosenberg F, Celikovic P, Michlmayr A, Leitner P, Dustdar S (2009) An end-to-end approach for QoS-aware service composition. In: Proceedings of the 13th IEEE international enterprise distributed object computing conference EDOC '09. IEEE Computer Society, Washington, pp 151–160

35. Studer R, Grimm S, Abecker A (2007) Semantic web services: concepts, technologies, and applications. Springer, Berlin

36. Thielscher M (1999) From situation calculus to fluent calculus: state update axioms as a solution to the inferential frame problem. Artif Intell 111(1–2):277–299

37. Thielscher M (2005) Reasoning robots: the art and science of programming robotic agents. Applied logic, vol 33. Springer, Netherlands

38. WSML Working Group (2008) The web service modeling language WSML. http://www.wsmo.org/wsml/wsml-syntax

39. Zeng L, Ngu AHH, Benatallah B, Podorozhny RM, Lei H (2008) Dynamic composition and optimization of web services. Distrib Parallel Databases 24(1–3):45–72