Bus Inf Syst Eng 60(5):393–413 (2018)
https://doi.org/10.1007/s12599-018-0554-0

RESEARCH PAPER

BoSDL: An Approach to Describe the Business Logic of Software Services in Domain-Specific Terms

Sebastian Schlauderer · Sven Overhage

Received: 1 November 2017 / Accepted: 8 July 2018 / Published online: 2 August 2018

Abstract Modular SaaS platforms that can flexibly be configured with software services, microservices, and the advent of the API economy provide new opportunities to realize even highly customized solutions in the cloud. The success of such endeavors depends on the ability of consumers to discriminate between offered services and choose those best fulfilling the requirements, though. To facilitate the assessment of services against functional requirements, this article proposes the Business-Oriented Service Description Language (BoSDL). It consists of: (1) a meta-model with rules to describe the business logic, that is, the functionality of a software service from a business-oriented perspective; (2) a textual presentation format based on English natural language; (3) a graphical notation based on the UML. Findings from a controlled experiment indicate that, compared to the state of the art, the information provided with the BoSDL enhances the ability of consumers to judge if software services satisfy existing functional requirements.

Keywords Software as a service · Service description · Service selection · Design science

Accepted after two revisions by the editors of the special issue.

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s12599-018-0554-0) contains supplementary material, which is available to authorized users.

Dr. S. Schlauderer (✉) · Prof. Dr. S. Overhage
Chair of Industrial Information Systems, University of Bamberg,
An der Weberei 5, 96047 Bamberg, Germany
e-mail: sebastian.schlauderer@uni-bamberg.de

Prof. Dr. S. Overhage
e-mail: sven.overhage@uni-bamberg.de

1 Introduction

The growing Software as a Service (SaaS) market provides enterprises with ever more opportunities to outsource even complex and customized software applications into the cloud (Rowsell-Jones et al. 2016). Promising opportunities particularly arise with the emergence of modular SaaS platforms like Salesforce, which can flexibly be configured and extended by selecting and composing software services with the desired functionality from providers of the surrounding ecosystem (e.g., Salesforce's AppExchange marketplace). By configuring and extending such modular SaaS platforms with services from a well-populated ecosystem, even strongly customized and industry-specific applications like enterprise resource planning systems can be realized in the cloud today. The possibilities to build highly customized SaaS systems on the basis of modular architectures might even further increase with the spreading of microservices – easily combinable software services that each provide a specialized capability like inventory management or billing (Pautasso et al. 2017) – and the advent of the API Economy, in which firms increasingly provide access to software capabilities and data that might be of value in service systems (Vukovic et al. 2016). Analysts emphasize that moving towards modular, flexibly configurable SaaS systems can bring several advantages for consumers such as a higher agility and increased scalability (IBM 2014; Herbert et al. 2016). Likewise, establishing modular SaaS systems can also benefit platform providers, for instance by extending the range of application of their platforms, and service providers, for instance by enabling them to efficiently offer services as parts of the SaaS system.

In general, service systems enable the (co-) creation of value through the particular configuration of the involved

actors and their resources (Vargo and Lusch 2004; Maglio et al. 2009). The success of modular, flexibly configurable SaaS systems accordingly depends on the consumers' ability to assess available services and compose those best fulfilling their requirements (Schlauderer and Overhage 2011; Polyviou et al. 2014; Sun et al. 2014). In well-populated service ecosystems, there likely exist several alternative software services that provide support for a specific task such as inventory management. Still, the provided functionality can vary substantially, both with respect to the business logic (e.g., different storage strategies might be used) and the service quality (e.g., different levels of availability might be achieved). Considering that complex enterprise solutions are composed of several different software services and that multiple candidates would have to be inspected in each case, an efficient assessment method is required. In such a scenario, especially approaches that support the selection of commercial-off-the-shelf (COTS) software cannot straightforwardly be applied due to their limited scalability. These approaches have been designed to facilitate the selection of a singular software product out of a small set of candidates. To obtain the information for the assessment of the candidates, they typically rely on in-depth evaluations of test versions and product catalogs (Kontio 1996; Land et al. 2008). If conducted for multiple software services, such a procedure would drastically increase the setup costs (Weyuker 2001).

To facilitate the setup of service systems, service description approaches have been emphasized in service science literature as a means to make external properties of provided services explicit rather than relying on their observation (Maglio et al. 2009; Ferrario et al. 2012). In the SaaS domain, several such approaches exist, allowing providers to specify relevant software service properties and to communicate them to potential consumers so that they can be compared to existing requirements (Ferrario et al. 2012; Terzidis et al. 2012). Interestingly, however, the vast majority of software service description approaches focuses on the specification of contractual terms (e.g., pricing models), quality of service levels, and the programming interface (Sun et al. 2014). For consumers, this information provides a basis to determine the eligibility of a software service from a commercial point of view, to assess the fulfillment of existing non-functional requirements, and to verify the technical compatibility of a software service. Yet, so far there only exist a few approaches with limited expressive power to specify the business logic, that is, the functionality of a software service from a business-oriented perspective (Sun et al. 2014). Therefore, it remains difficult for consumers to determine whether a software service also fulfills existing functional requirements, although this actually is the most important selection criterion (Repschlaeger et al. 2012; Polyviou et al. 2014). Studies of leading SaaS marketplaces confirm that especially the information provided with respect to the business logic of software services does not suffice to support an efficient assessment and selection (Hrach and Alt 2018; Schlauderer and Overhage 2011). As far as functional requirements are concerned, consumers are hence often required to treat software services like experience goods, whose appropriateness becomes entirely clear only after intensive use and observation.

To fill this literature gap and better support the assessment and comparison of software services against functional requirements, we propose the Business-Oriented Service Description Language (BoSDL). It has been designed as a lightweight approach to specify the business logic, that is, the functionality of a software service from a business-oriented perspective. With the design of the BoSDL, we examine the following research questions: *How can the business logic of software services be specified in domain-specific terms? How does a specification of the business logic in domain-specific terms affect the assessment and selection of software services?* The resulting BoSDL consists of three elements: (1) a theoretically grounded meta-model with rules to describe the business logic of a software service as system of domain-specific concepts; (2) a textual format to specify the business logic in a standardized form of English natural language; (3) a graphical format to depict the business logic in a two-dimensional way. Construction of the BoSDL is based on the design science research (DSR) approach (Iivari 2007; Hevner et al. 2004; Gregor and Hevner 2013). Established DSR principles particularly demand to evaluate the usefulness of the developed artifact in its intended context (Gregor and Hevner 2013). We therefore chose to examine the usefulness of the BoSDL for the assessment and selection of software services in a controlled experiment with 126 participants.

The results of our research contribute to the emerging service systems engineering (SSE) discipline, which "seeks to advance knowledge on models, methods, and artifacts that enable or support the engineering of service systems" (Böhmann et al. 2014). In particular, we provide novel evidence-based knowledge on the design of service description languages, which are frequently emphasized in literature as a means to make service properties explicit that are relevant for the configuration of service systems (Maglio et al. 2009; Ferrario et al. 2012). In this respect, the presented BoSDL provides a nascent design theory with operational principles (Gregor and Hevner 2013) governing the description of the business logic of software services, an area that has not been addressed sufficiently yet (Polyviou et al. 2014; Sun et al. 2014). In the SaaS domain, the BoSDL complements existing description approaches, which focus more on describing the quality, the

programming interface, and/or contractual properties of software services (Polyviou et al. 2014; Sun et al. 2014). While we designed the BoSDL specifically with software services in mind, the resulting description remains independent of their software-technical realization. The BoSDL might hence even be generalizable to more precisely describe the business logic of services outside the SaaS domain.

The remaining presentation follows established guidelines for the publication of DSR studies (Gregor and Hevner 2013): in Sect. 2, we discuss related work and the purpose of our approach in more detail. Section 3 elaborates on the specific design science research approach that was adopted in our study. In Sect. 4, we introduce the BoSDL and its constituents. Section 5 describes the evaluations conducted to demonstrate the feasibility and usefulness of the BoSDL. In Sect. 6, we discuss the research contributions, implications for academia and practice, and future research directions.

2 Background and Related Work

Successful SaaS platforms that can flexibly be configured with services from well-populated provider ecosystems, the emergence of microservices, and the prophesied advent of the API Economy open up promising opportunities to realize even customized enterprise applications in the cloud. From a SSE perspective, however, the trend towards SaaS systems with modular, flexibly configurable architectures also poses new challenges. Generally, it appears still necessary "to enhance the possibilities for modularization, standardization, contextualization and re-configuration of service components" (Böhmann et al. 2014). An important measure to better support the design of modular, flexibly configurable SaaS systems is to enable service consumers to efficiently discriminate between alternative services and select the ones best fulfilling their requirements (Turner et al. 2003).

To facilitate an assessment and matchmaking against existing requirements, the importance of software descriptions as an equivalent to product descriptions in other engineering disciplines has been emphasized repeatedly in software engineering literature. A software description describes the functionality of a software artifact by specifying "in precise terms the intended effect of a piece of software" (Gehani and McGettrick 1986). With service descriptions, the SSE discipline has adopted a comparable concept to specify the service functionality and help matching the provider's offering to the consumer's needs (Turner et al. 2003). Especially in the context of modular SaaS systems, which are composed of multiple software services, service descriptions are discussed as a crucial success factor to facilitate the assessment and configuration process (Turner et al. 2003; Repschlaeger et al. 2012).

2.1 Related Work

Basically, a software service can be regarded as an offered functionality (or set of functionalities) that consumers can invoke to support a business task. The functionality of a software service is determined by its implementation and offered by means of an interface, which defines one or more accessible operations as well as the information that is exchanged when an operation is invoked by a consumer. To ascertain if a software service is able to support a particular business scenario, the consumer has to assess the provided functionality against his/her requirements. In general – and in compliance with reference architectures such as the Architecture of Integrated Information Systems (ARIS, Scheer 2000) –, the functionality of a software service can be expressed on three abstraction layers (Turner et al. 2003; O'Sullivan 2006): the *conceptual layer* expresses the business logic, that is, the provided functionality from a business-oriented perspective; the *quality layer* expresses the service quality, that is, the functionality from a performance-oriented perspective; the *technical layer* expresses the programming interface, that is, the functionality from a programmatic perspective. The description of the service functionality is often complemented with a specification of commercial characteristics such as the terms of pricing and contract (Barros and Oberle 2012).

To determine in how far the different abstraction layers are taken into consideration by existing service specification and matchmaking approaches, we conducted a literature search following the recommendations given by Webster and Watson (2002). In particular, we queried the AIS Electronic Library, IEEE Digital Library, ACM Digital Library, EBSO Host, and Google Scholar using keywords such as "description", "specification", "quality", "functionality", or "semantics" together with "service" and "software". The results of our literature analysis indicate that the vast majority of service description and matchmaking approaches concentrates on properties of the technical and quality layers. For instance, several established approaches such as the Web Service Description Language (WSDL) or the Web Ontology Language for Services (Chinnici et al. 2007; Martin et al. 2005) only support the specification of the methods and data types of the programming interface. Regarding the technical layer, there furthermore exists a whole plethora of approaches that can be used to specify additional elements of the programming interface like pre- and post-conditions, notifications, or transactions (Akkiraju et al. 2005; Graham

et al. 2006; Cabrera et al. 2002). We also found several approaches that focus on the specification of the quality of software services. As part of the so-called WS-* languages, for instance, there exist approaches to specify individual quality attributes such as the security and reliability of software services (Atkinson et al. 2002; Iwasa 2004). In addition, we found approaches such as the Web Service Modeling Ontology and the closely related Description of Service Capabilities and Properties (DSCP), which support the detailed annotation of methods of the programming interface with quality attributes (Roman et al. 2005; O'Sullivan 2006), for instance to describe their performance or reliability. Building upon such a description, several approaches have been proposed to support an automated matching of non-functional requirements to service quality profiles (e.g., Wang et al. 2006). There also exist approaches for the management of service level agreements such as the Web Service Level Agreement framework (Ludwig et al. 2003), which allows providers to formulate and monitor quality of service levels.

In comparison to the before-mentioned approaches, we discovered only a few service description approaches that support a specification of the business logic, though. As part of an approach to specify the capabilities of software services, Oaks et al. (2003) propose to annotate each method and parameter of the programming interface with a business term (i.e., a verb or a noun) in order to characterize their conceptual semantics. They also suggest the specification of synonyms and the placement of links to external definitions of the terms. Taking a similar approach to describe the business logic of software artifacts, Vitharana et al. (2003) suggest augmenting the description of the programming interface with business terms to express the business meaning of each method and parameter. Both approaches support a rudimentary description of the service logic, that is, the provided functionality from a business perspective. However, the expressive power is limited as each method or parameter can only be characterized by a single business term. It is hence not possible to specify the business logic in detail, since this would require setting multiple business terms into relation (e.g., to express that a "storage strategy" parameter is applied to an entire "warehouse area"). Such details can only be provided as part of a definition of the business term in natural language. Despite this limitation, especially the service capability description as suggested by Oaks et al. (2003) has been taken up in other service description approaches. The DSCP approach (O'Sullivan 2006) uses it as a basis to specify quality characteristics of software services. The concepts to describe the business logic, however, remain unchanged since the approach is "not attempting to provide a functional description of a service" (O'Sullivan 2006). The Business Service Description Language (Le et al. 2010) also builds upon the service capability description as introduced by Oaks et al. (2003) and adds "concepts that describe the decomposition of business services and some non-functional properties". As the language rather focuses on supporting the decomposition of business services, the expressive power with respect to the business logic again remains unchanged.

During our literature analysis, we also examined holistic approaches, which aim at providing a complete service description and hence should cover all three abstraction layers. To document the functionality of software services from a business-oriented perspective, the Universal Description, Discovery, and Integration standard introduces a faceted classification with facet-value pairs such as "application domain: inventory management" as part of its yellow pages (UDDI 2002). Besides a simple classification of the application domain, however, no statements about the business logic can be expressed. While the programming interface can be formally specified on the green pages, a detailed description of the business logic could only be included using free-text fields and natural language. More complex statements regarding the business logic can be expressed with the Unified Service Description Language, which explicitly supports the specification of capabilities to describe the supported business tasks as part of its Functional Module (Barros and Oberle 2012). In the USDL, a capability is defined as a publicly visible function, which can have parameters and can be decomposed into sub-functions. Functions and parameters can each be annotated with a description, which can refer to a business term that is defined in an external ontology. The expressive power is hence similar to the service capability description as introduced by Oaks et al. (2003). While it is possible to specify composition relationships between capabilities (e.g., "commissioning articles" consists of "executing a picking plan" and "updating stock levels") in the USDL, other forms of complex business logic still cannot be expressed.

In summary, we hence found the support to describe the business logic of software services to be limited. This observation is corroborated by literature studies (Polyviou et al. 2014; Sun et al. 2014) and surveys of leading SaaS marketplaces, which confirm that the information provided with respect to the business logic does not suffice to support an efficient assessment and selection of software services (Hrach and Alt 2018; Schlauderer and Overhage 2011). Although the fulfillment of functional requirements actually is the most important selection criterion (Repschlaeger et al. 2012; Polyviou et al. 2014), its evaluation appears to be least supported by today's service description approaches. Admittedly, the business logic of software services might be documented using comprehensive enterprise modeling frameworks such as ARIS or the

Multi-Perspective Enterprise Modeling framework (MEMO, Frank 2014). Such frameworks typically make use of multiple languages and views to document business landscapes, however. It would hence also be necessary to use several languages to describe the business logic of software services (in particular, simply using a business process modeling notation would not suffice as we will show in Sect. 4). Such frameworks would furthermore have to be used in a strongly restricted fashion to achieve comparable descriptions. As no such approach exists, we plead for the development of a dedicated, lightweight service description language, which draws from such approaches but is tailor-made to specify the business logic of software services and can be integrated into existing approaches such as the USDL.

2.2 Problem Statement

To identify an eligible software service out of a set of candidates, consumers have to assess whether they fulfill the functional, non-functional, and technical requirements of the particular application scenario (Kontio 1996). In general, all three abstraction levels of the service functionality hence need to be analyzed. As service description, today's SaaS marketplaces typically provide a general-purpose description in natural language and optionally available formal specifications, which specify quality attributes and/or the programming interface of the service, for instance using the WSDL (Hrach and Alt 2018). Without a precise description of the business logic – i.e., the functionality from a business-oriented perspective –, it remains difficult to determine if a software service fulfills the existing functional requirements, however. We illustrate this problem by referring to the case of a large German supermarket company that wanted to identify suitable software services to configure a cloud-based enterprise resource planning system using a leading SaaS platform. Apart from the SaaS platform, the initial system was comprised of several different software services that were chosen from the accompanying marketplace. In the following, we limit the discussion to the process of identifying a suitable inventory management service. Table 1 shows an excerpt of the company's functional requirements. The content is denoted in natural language but comparable to that contained in more formal specifications.

To evaluate if a service candidate fulfills the functional requirements, its general-purpose description was examined at first. Table 2 depicts a fragment of a typical general-purpose description for a service candidate that was found on the marketplace. The description contains an explanation of the functionality in business terms, but is obviously neither detailed nor precise enough to assess if the functional requirements depicted in Table 1 are fulfilled. The company therefore had to try inferring the required information from the available programming interface specification. Table 3 shows a simplified excerpt of the WSDL specification of the same service candidate to illustrate the process. The complete specification contains 1501 lines of code in Extensible Markup Language. It encompasses 31 operations and 25 data types, which had to be analyzed to evaluate whether the service candidate fulfills the functional requirements.

Most of the business logic nevertheless needed to be inferred from the specification. For instance, requirement ③ demands support for the management of warehouses, in which the storage strategy varies from area to area (Table 1). From line 20 of the specification, consumers can infer that the strategy can even be configured per storage

Table 1 Exemplary functional requirements document (excerpt)

We want to use a network-available service to manage our warehouses with up to 5000 storage cells ⓪. Currently, we operate seven warehouses ①, each of which is subdivided into receiving, storage, picking, and dispatching areas ②. Depending on the area, either a fixed-bin or chaotic storage strategy is applied ③. Furthermore, a first-expires-first-out (FEFO) commissioning strategy is applied for goods in the storage area ✪. Each area consists of a number of storage bins, which may have varying sizes and tonnage capacities ④. A storage bin is located by the numbers of its corridor, rack, and level ⑤. Moreover, storage bins are suited to contain a certain number of either identical articles or standardized pallets ⑥. Standardized pallets, especially those to be shipped to the various stores, can contain up to 20 different articles ⑦. The warehouse management service has to be able to create, read, update, and delete warehouse structures ⑧. It is required to automatically calculate plans for picking articles from the warehouse on the basis of customer orders and for distributing articles to storage bins on the basis of delivery notes ⑨. The stock level of articles has to be managed by accounting services ⑩

Table 2 General-purpose description of a service candidate (excerpt)

This package provides inventory management with multi-warehouse inventory capabilities, inventory batching from fixed-bin and chaotic storage, thresholds for alerts when inventory is low, and advanced inventory status management options. The tool allows you to configure when you want to reserve and commit from inventory and at what level you would like an alert to low quantities of your products. It also supports inventory tracking in multiple warehouse locations as well as inventory adjustment and transfer. For more information check out our free trial version and the video documentation here

Table 3 Simplified WSDL excerpt of a service candidate

```
01 <wsdl:definitions>
02   <wsdl:types>
03    <s:schema targetNamespace="http://www.sample.storage-management.org/">
04     <s:element name="StructureStorage"><s:complexType><s:sequence>
05        <s:element minOccurs="1" maxOccurs="1" name="name" type="s:string"/>
06        <s:element minOccurs="1" maxOccurs="unbounded" name="storageareas" type="StructureStorageArea"/>
07     </s:sequence></s:complexType></s:element>
08     <s:element name="StructureStorageArea"><s:complexType><s:sequence>
09        <s:element minOccurs="1" maxOccurs="1" name="type" type="tns:AreaType"/>
10        <s:element minOccurs="1" maxOccurs="750000" name="bins" type="Bin"/>
11     </s:sequence></s:complexType></s:element>
12     <s:simpleType name="AreaType"><s:restriction base="s:string">
13        <s:enumeration value="storing"/><s:enumeration value="picking"/><s:enumeration value="shipping"/>
14     </s:restriction></s:simpleType>
15     <s:element name="Bin"><s:complexType><s:sequence >
16        <s:element minOccurs="0" maxOccurs="1" name="typeId" type="s:string"/>
17        <s:element minOccurs="1" maxOccurs="1" name="slot" type="s:int"/>
18        <s:element minOccurs="1" maxOccurs="1" name="level" type="s:int"/>
19        <s:element minOccurs="0" maxOccurs="10" name="articleCounts" type="ArticleCount"/>
20        <s:element minOccurs="1" maxOccurs="1" name="strategy" type="tns:Strategy"/>
21     </s:sequence></s:complexType></s:element>
22     <s:element name="ArticleCount"><s:complexType><s:sequence>
23        <s:element minOccurs="1" maxOccurs="1" name="id" type="s:string"/>
24        <s:element minOccurs="1" maxOccurs="1" name="count" type="s:int"/>
25     </s:sequence></s:complexType></s:element>
26     <s:simpleType name="Strategy"><s:restriction base="s:string"><s:enumeration value="static"/><s:enumeration value="chaotic"/>
27     </s:restriction></s:simpleType>
28     <s:element name="getStorageIds"><s:complexType><s:sequence>
29        <s:element minOccurs="0" maxOccurs="5" name="getStorageIds" type="s:string"/>
30     </s:sequence></s:complexType></s:element>
31    </s:schema>
32   </wsdl:types>
33   <wsdl:message name="createStorageSoapIn"><wsdl:part name="parameters" element="tns:StructureStorage"/></wsdl:message >
34   <wsdl:message name="createStorageSoapOut"><wsdl:part name="parameters" element="tns:createStorageResponse"/></wsdl:message>
35   <wsdl:message name="getStorageIdsSoapIn"><wsdl:part name="parameters" element="tns:getStorageIdsInput"/></wsdl:message>
36   <wsdl:message name="getStorageIdsSoapOut"><wsdl:part name="parameters" element="tns:getStorageIds"/></wsdl:message>
37   <wsdl:portType name="Storage Management Services">
38    <wsdl:operation name="createStorage"><wsdl:documentation>Role: Storage Management. Description: Uses a storage structure
39     to create a new storage. Inputs: datatype StructureStorage. Output: new storage ID.</wsdl:documentation>
40     <wsdl:input message="tns:createStorageSoapIn"/><wsdl:output message="tns:createStorageSoapOut"/>
41    </wsdl:operation>
42    <wsdl:operation name="getStorageIds"><wsdl:documentation>Role: Storage Management. Description: Gets IDs of existing storages.
43     Output: storage IDs (arraySize <= 5).</wsdl:documentation>
44     <wsdl:input message="tns:getStorageIdsSoapIn"/><wsdl:output message="tns:getStorageIdsSoapOut"/>
45    </wsdl:operation>
46   </wsdl:portType>
47 </wsdl:definitions>
```

bin, thus fulfilling the requirement. In contradiction to requirement ⑦, however, line 19 suggests that storage bins support a maximum of 10 different articles only. The service also does not seem to allow the modeling of separate areas for receiving and dispatching, thus breaching requirement ②. Instead, it supports a more general concept of shipping areas according to line 13. In violation of requirement ①, line 29 implies that a maximum of five warehouses is supported as the operation "getStorageIds" returns an array with a capacity of five. The commissioning strategy obviously cannot be customized using the interface and is not mentioned at all. It hence remains unclear if the service supports a FEFO strategy as demanded by requirement ✪.

Since inferences from the programming interface description involve a significant amount of interpretation and do not allow the evaluation of all functional requirements, consumers cannot be sure about the implemented business logic without obtaining additional information. We hence posit that a service description should contain an explicit documentation of the provided functionality from a business-oriented perspective to better support an assessment against functional requirements. To achieve this goal, *a service description language needs to document the business logic of all operations accessible on the interface* (*design objective*).

2.3 Solution Requirements and Expected Benefits

Besides the design objective, we identified six additional requirements that the design of the service description language ought to fulfill in two steps. In a first step, we surveyed extant conceptual modeling and knowledge representation literature for general, methodological language requirements. In a second step, we identified application-specific requirements that were found to be relevant particularly in the intended usage context of the service description language. To identify such requirements, we examined a real-world development project, in which software services were assessed and selected based on functional requirements (Sect. 2.2). We applied two established elicitation methods (Pohl 2010; Zowghi and Coulin 2005): first, we observed the project to study applied practices and techniques. Afterwards, we interviewed involved system designers and domain experts to gather their critique and expectations of the supporting service description language. To get an overview of the key points, we applied open coding techniques to the interview transcripts. We then grouped statements of similar topic to identify consistently articulated requirements. Finally, we cross-verified our results and impressions with reports from literature.

Surveying literature led to three methodological requirements (R1–R3): Generally, a language should be able to represent all relevant concepts of the universe of discourse in order to work as an effective means of communication (Wand and Weber 1993; Davis et al. 1993; Shanks et al. 2003). Although it is difficult to formally prove, we accordingly formulate requirement *R1 (completeness)*: the service description language has to cover all concepts necessary to specify the business logic of the operations available on the interface. In particular, its expressive power needs to be adequate to describe the business logic of software services across many domains.

At the same time, a language needs to avoid any overload, redundancy, or excess of its constructs (Goodman 1976; Wand and Weber 1993; Moody 2009). A language construct is overloaded if it maps to more than one concept in the universe of discourse. It is redundant, if there is another construct mapping to the same concept, and excessive, if it maps to no concept at all. As the presence of such defects makes the usage of a language difficult, we formulate *R2 (clarity)*: the service description language needs to provide non-redundant and unambiguous constructs.

In principle, a specification language also should establish strict guidelines on how to use its constructs (Davis et al. 1993; Ortner and Schienmann 1996; Moody 2009). Such guidelines do not only facilitate its use. They also help reducing variations among service descriptions, making them more comparable. We therefore posit *R3 (strictness)*: the service description language needs to establish normative rules to describe the business logic.

As an effective measure to fulfill R2 and R3, literature on the design of notations recommends introducing a formal language basis (Davis et al. 1993). It hence appears appropriate to call for a formal basis of the aspired service description language. Such a formal language would moreover provide better support for automated processing, validation, and reasoning (Davis et al. 1993).

Analyzing the gathered project data led to three application-specific requirements (R4–R6): Both our observations and recent literature suggest that the assessment and selection of software services in practice still is a considerably manual process (Eisa et al. 2016). The interviewed domain experts and system designers consistently mandated that the service description language needs to be straightforwardly understandable by all stakeholders. Therefore, we formulate *R4 (understandability)*: the service description language needs to introduce an understandable presentation format with constructs that prospective users are familiar with.

The majority of the interviewed stakeholders additionally uttered a demand for a lightweight, economical language that can be used efficiently during the evaluation stage. In particular, they emphasized that service descriptions have to be easier to evaluate than test versions of software services. As literature also provides indications that a parsimonious language design might facilitate its acceptance in practice (Moody 2009), we posit *R5 (simplicity)*: the service description language should introduce a compact set of constructs.

Several interviewees also mandated that the business logic should be described independently of implementation concerns. They found a description, which is rooted in implementation details, to unnecessarily intermix aspects of the conceptual business logic with aspects of its technological realization. For instance, line 29 of the WSDL file shown in Table 3 states $<$`s : element` *minOccurs* $=$ *″0″ maxOccurs* $=$ *″5″ name* $=$*″getStorageIds″ type* $=$ *″s : string″*/$>$. Without reference to its technological realization, the business logic could be formulated in a more accentuated way: “the number of manageable warehouses is limited to a maximum of 5”. As technological platforms furthermore differ between SaaS ecosystems and technology-dependent descriptions of the business logic were hence found difficult to compare, we formulate *R6 (technology independence)*: the service description language should express the business logic independently of its technological realization. Note that a mapping between both aspects was still considered helpful to understand, which parts of the programming interface implement a particular part of the business logic.

All in all, we identified six design requirements for the aspired service description language. Interestingly, other service description approaches seem to be based on comparable requirements, thus lending additional support to their importance (O’Sullivan 2006). Referring back to its intended usage context, we expect a language that fulfills the above-mentioned design objective and requirements to *enhance the usability of service descriptions for service consumers during the assessment stage*. Generally, the usability of an artifact is determined by three aspects (Frøkjær et al. 2000): the effectiveness (i.e., accuracy), efficiency (i.e., effectiveness in relation to effort), and satisfaction (i.e., users comfort). We consequently formulate the following expectations when introducing a language to specify the business logic of software services as propositions:

- *P1 (Higher effectiveness)* We expect the ability of consumers to identify eligible services to increase.
- *P2 (Higher efficiency)* We expect the time needed by consumers to identify eligible services to drop.
- *P3 (Higher satisfaction)* We expect the satisfaction of consumers with the service description to increase.

3 Research Method

To fill the identified literature gap and reach the design objective, we propose the BoSDL as a new approach to specify the business logic of software services. Its construction is based on the DSR approach that provides principles and guidelines to ensure the rigorous, scientific development of innovative artifacts to solve relevant problems (Iivari 2007; Hevner et al. 2004; Gregor and Hevner 2013). Generally, such artifacts may be constructs, methods, models, or instantiations (Hevner et al. 2004). In our case, the developed artifact is a *method* to describe the business logic of software services. The method engineering discipline provides different conceptualizations of the term. In our DSR study, we adopted the following conceptualization (Goldkuhl et al. 1998): a method consists of a *concept*, which defines what aspects of the reality are relevant and should be captured, a *procedure*, which describes how to capture these aspects, and a *presentation format* to document the results. We hence centered our DSR study around these method elements.

The design of the BoSDL is informed by theories of the conceptual modeling discipline, which is concerned with "the activity of formally describing some aspects of the physical and social world around us for the purposes of understanding and communication" (Mylopoulos 1992). From a method engineering perspective, the BoSDL was designed as a special-purpose (or domain-specific) conceptual modeling language (Frank 2013), which is specialized on describing the business logic of software services. With respect to the DSR knowledge contribution framework (Gregor and Hevner 2013), our research goal thereby was to create a new solution to a known problem in form of a more effective and efficient technique to describe the business logic.

To ensure a rigorous and traceable design procedure, we adopted the DSR process suggested by Peffers et al. (2007) as macro-process for our study. Its iterative nature allowed us to incorporate experiences gathered during earlier demonstration and evaluation stages (see Fig. 1). It hence gave us the opportunity to constantly refine the BoSDL. Following our research goal, we furthermore decided to follow extant guidelines for the design of special-purpose conceptual modeling languages during the solution design and development. We therefore incorporated two processes, which were proposed by Frank (2013) to specify the language concept and to design the presentation format, as micro-processes into our procedure. All in all, we proceeded as follows: During the first stage of the DSR process, we defined the research problem and the purpose of the BoSDL. Following recommendations of Peffers et al. (2007) and Sonnenberg and vom Brocke (2012), we conducted reviews of the literature and practitioner activities

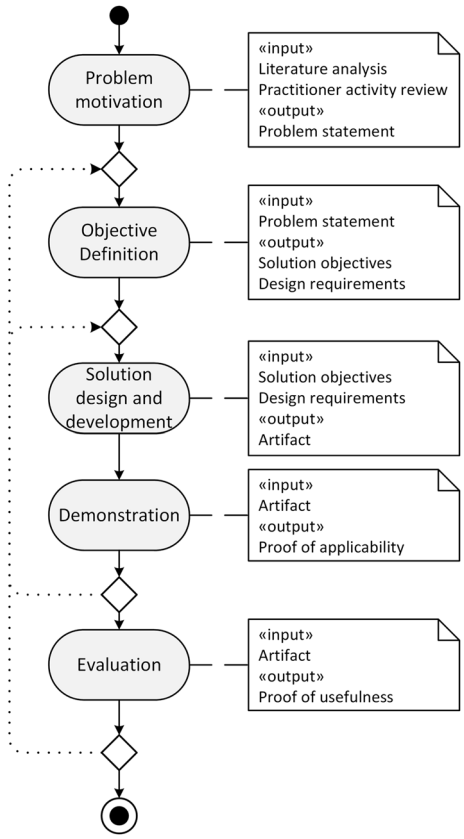

Fig. 1 Design cycle, based on Peffers et al. (2007)

as ex-ante evaluations to confirm the relevance of the problem statement and the novelty of the approach (Sect. 2). Based on the gathered findings, we defined the design objective and derived design requirements in the second stage. In the third stage, we designed the solution concept. Following our micro-process, we developed and theoretically justified a meta-model as the language concept. It determines how the business logic of software services should be expressed. We then implemented the solution by developing presentation formats on the basis of the meta-model. In the last two stages, we focused on conducting ex-post evaluations of the BoSDL (Sonnenberg and vom Brocke 2012). In the fourth stage, we examined the applicability of the BoSDL to specify the business logic of various software services. In the fifth stage, we conducted laboratory experiments to examine the usefulness of the BoSDL in its intended usage context, that is, during the assessment and selection of software services.

Overall, we conducted four iterations of the DSR process. The first two iterations ended after the fourth stage, because our attempts to specify complex software services indicated that we were not yet able to express all relevant content. We hence decided to refine the language concept (i.e., the meta-model). During the third iteration, we

arrived at a state in which all relevant content could be described. We demonstrated the technical feasibility of the BoSDL by creating a modeling tool as a proof of concept. Thereafter, we conducted a laboratory experiment to evaluate its usefulness. During the course of the evaluation, we observed that the generic association relationship, which was part of the language to interrelate information objects, was defined too broadly and often used instead of specialization and aggregation relationships. After consulting extant conceptual modeling literature (e.g., Ortner and Schienmann 1996), we decided to drop this relationship type and introduced an object connection relationship to fill the gap. Since participants of the laboratory experiment moreover indicated to prefer a graphical presentation format, we decided to develop such an additional format. During the fourth iteration, we furthermore implemented a tool that is able to convert between the two language formats and conducted a laboratory experiment with a larger number of participants (Sect. 5.2).

4 Business-Oriented Service Description Language

The BoSDL documents the business logic of software services as a lightweight conceptual model. Using a conceptual model as means for documentation was deemed appropriate since a conceptual model is a proven instrument to provide "an accurate, complete representation of someone's or some group's perceptions of the semantics underlying a domain or some part of a domain" (Bodart et al. 2001). The contents of a conceptual model are described independently of programming technologies and implementation concerns (Hadar and Soffer 2006; Topi and Ramesh 2002), thus allowing the BoSDL to focus on describing the functionality of software services from a business-oriented perspective.

4.1 Concept

Following recommendations to design special-purpose modeling languages (Frank 2013), we formally specified the language concept as a meta-model using the Unified Modeling Language (UML) as notation. The core concept of the BoSDL is based upon the insight that the semantics of a domain is expressed by its particular system of technical terms (Bunge 1977), in our case business terms. The intent of the BoSDL accordingly is to specify a system of concepts to express the business logic of software services. Such a system of concepts can be documented by two measures (Ortner and Schienmann 1996; Gómez-Pérez et al. 2004): the building of (1) a *vocabulary*, which contains the domain-specific business terms, and (2) a set of *statements*, which interrelate business terms to express complex semantics. In the inventory management domain, characteristic terms would be "storage bin", "storage strategy" etc. Statements to express complex semantics would be "a warehouse consists of multiple storage bins" or "a warehouse area is operated according to a storage strategy".

To specify the language concept, we have to identify and reconstruct the types of concepts and relationships, which are required to describe the business logic of software services (Frank 2013). Following our design objective, we aim at documenting the business semantics of all methods accessible on the programming interface of a software service. The programming interface of a software service basically consists of *ports*, *methods*, and *data types* (Chinnici et al. 2007). From a conceptual, business-oriented perspective, a data type is a technical representation of a piece of information, which we refer to as *information object*. An interface method is the result of a (partial) automation of a business task, which we refer to as *function*. A port groups thematically related interface methods. Conceptually, such a grouping of methods corresponds to the (partial) automation of a composite business task that can again be expressed as function. However, a composite task usually also imposes some meaningful course of action, in which its constituent tasks are meant to be executed. We therefore decided to objectify the course of action and refer to it as *process*. A software service supporting the inventory management may, among others, process information objects like purchase orders, provide functions to generate and execute picking plans, and implement complex processes such as stock placement and removal.

During a thorough analysis of the universe of discourse, all three concept types were found to be relevant to express the business logic of software services. We hence included them into the meta-model where they determine the content of the vocabulary (see Fig. 2). Information objects can be mapped onto data type definitions or parameters of the programming interface, functions can be mapped onto interface methods, and processes can be mapped to ports. However, the vocabulary will likely also contain business terms not directly related to the programming interface. Such terms are used to describe the business logic of software services in context. For each business term, the name and an ID has to be specified. Short and long definitions to specify its meaning can be added. Note that concept types such as the organizational unit were not reconstructed as language elements as such contexts can vary considerably when reusing a software service and should hence not be predetermined.

To express complex aspects of the business logic, statements can be formulated based on the vocabulary. When analyzing the universe of discourse, we identified

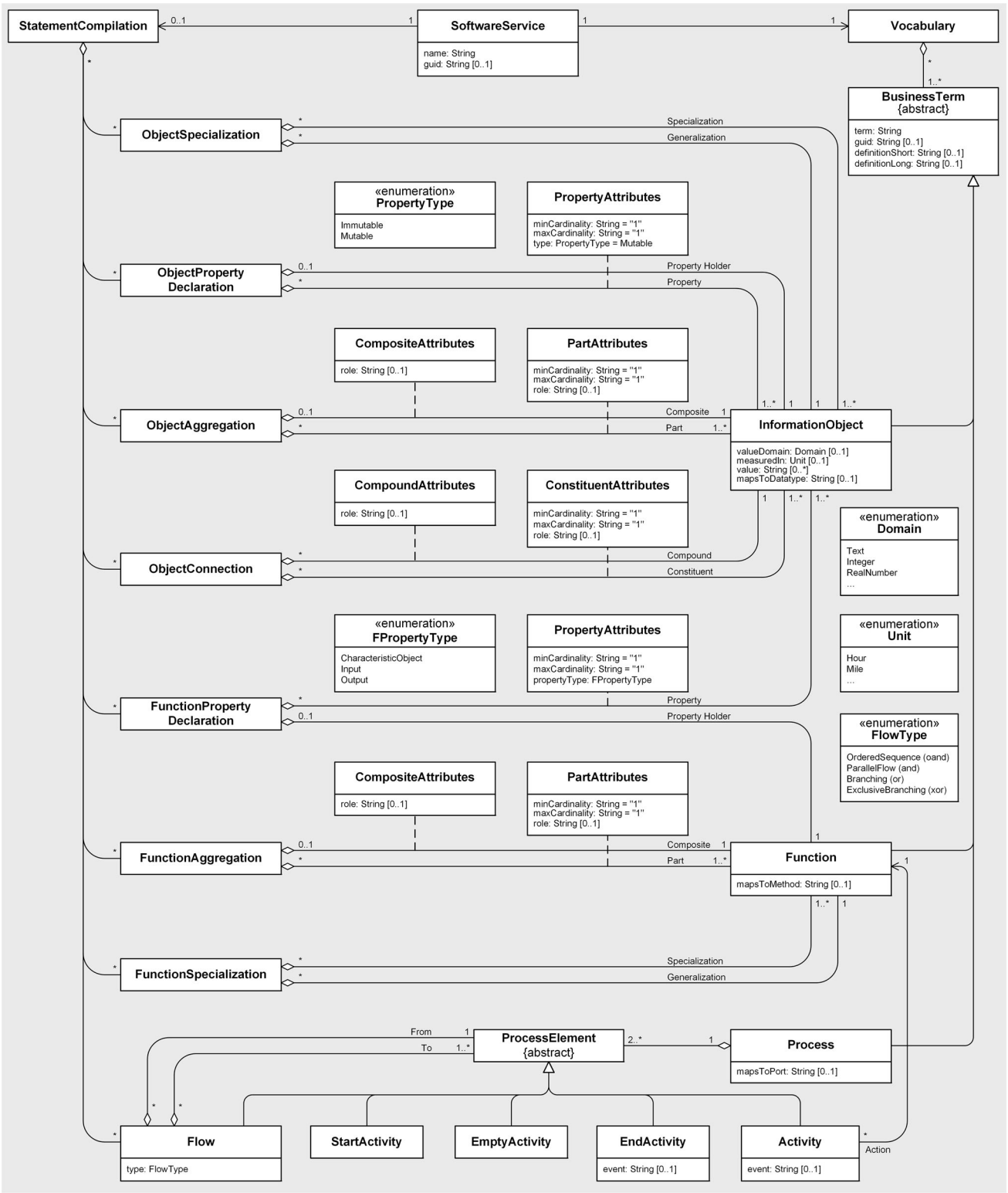

Fig. 2 BoSDL meta-model with language constructs (depicted in UML)

several different formulations to express conceptual relationships. As we found it difficult to identify patterns for different kinds of relationships, we decided to introduce relationship types based on findings published in the conceptual modeling literature. Ortner and Schienmann (1996) show that four basic types of relationships exist to

interrelate *information objects*: *specializations*, *property declarations*, *aggregations*, and *connections*. Our analysis showed that the suggested relationship types were able to subsume the various instances of relationships that we had encountered. We hence decided to include the relationship types into the meta-model (see Fig. 2).

An *object specialization* expresses a conceptual subsidiarity between information objects, which means that one is more specific than the other (Ortner and Schienmann 1996). Used repeatedly, they constitute abstraction hierarchies of information objects. To describe inventory management functionality, we can for instance specify: "a picking area is a special warehouse area".

An *object property declaration* assigns one information object as dependent part to another (Ortner and Schienmann 1996). An exemplary statement to describe the business logic of an inventory management service would be: "a storage bin has a loading capacity". Properties have simple values, which are specified as enumerations or by appointing a domain and an optional measurement unit: "a storage strategy is either fixed-bin or chaotic", "a loading capacity is a real number measured in pounds". Therefore, the meta-model contains a set of domains and measuring units (see Fig. 2).

An *object aggregation* combines information objects to form composite objects with emergent properties (Ortner and Schienmann 1996). It combines information objects that stand for themselves. Characterizing an inventory management service, we could specify: "a warehouse consists of one or more warehouse areas".

An *object connection* objectifies the relationship between its constituent objects and allows to view it as an information object in its own sense (Ortner and Schienmann 1996). To describe an inventory management service, we could specify: "a picking plan position connects articles and their storage locations".

Conceptual modeling literature shows that functions can be connected to each other and to information objects by three kinds of relationships (Ortner and Schienmann 1996; Scheer 2000): *specializations*, *property declarations*, and *aggregations*. As we found all relationship types to be relevant in our universe of discourse, we included them into the meta-model (see Fig. 2).

Function specializations and aggregations have a meaning comparable to that of their counterparts for information objects. Specializations express a conceptual subsidiarity between functions, meaning that a function is more specific than the other (Ortner and Schienmann 1996). For an inventory management service, we could specify: "batch picking is a special form of commissioning articles". Aggregations combine functions to form a composite (Ortner and Schienmann 1996). Similar to the USDL, they can be used to refine complex business functions into constituents, which are implemented on the interface (Barros and Oberle 2012). To describe the business logic of an inventory management service, we could specify: "article commissioning consists of executing a picking plan and updating a stock level".

Function property declarations assign information objects to functions as inputs or outputs (Ortner and Schienmann 1996). To characterize the business logic of an inventory management service, we could specify: "a user commissions" (function) "one or more articles" (characteristic object/input) "with a picking plan" (input) "to a shipment" (output). Naturally, functions operate on a characteristic object, which is already part of the function term (i.e., "commission article" instead of "commission"). With a property declaration, however, details such as a cardinality can be added.

Aggregations between functions express a static part-whole relationship. As the temporal order, in which functions should be carried out, also played an important role during our analysis of the universe of discourse, we decided to provide support for its specification. We therefore included a *flow* relationship type into the meta-model (see Fig. 2). Note that this relationship is not intended to describe a business process, in which the service can participate. As software services ought to be reusable in different contexts (and hence business processes), such an intent would be counterproductive. Instead, the flow relationship is meant to express temporal constraints that exist between supported business functions. They can be expressed using simple workflow patterns such as sequences, parallel flows, and branches (Aalst et al. 2003). To characterize an inventory management service, we could specify: "executing a commissioning plan precedes updating the stock level".

Figure 2 depicts the resulting meta-model. It contains an abstract syntax consisting of three types of business terms and eight relationship types to specify the business logic of software services from a conceptual perspective. Note that several constraints required to ensure language consistency are not depicted.

4.2 Presentation Formats

The BoSDL proposes a textual and a graphical presentation format to specify the business logic of software services. Initially, we decided to propose a textual format based on the English natural language, because we wanted designers and business users to be able to straightforwardly read and understand the specified business logic. Human understandability was deemed to be particularly important, since the assessment of software services still is a considerably manual process.

The textual format is based on the English natural language, but introduces standardized sentence patterns. They conform to the contents of the meta-model and allow describing the business logic of software services as human-readable statements (see Fig. 3, left). Compared to natural language, the proposed format is highly restrictive as it only allows statements conforming to the sentence patterns. Using such a normative language approach leads to more precise and homogeneous specifications as the semantics of all statements is clearly defined (Ortner and Schienmann 1996). We therefore expect it to facilitate a correct understanding of the specified business logic. Furthermore, we are able to use standardized specifications for automated processing and compilation (see online appendix, available online via http://link.springer.com).

During the course of our study, we received feedback that some stakeholder groups might prefer a graphical presentation format, however. We therefore decided to develop such a format based on the UML, which we adapted by creating a profile. It provides symbols and rules for composition to depict the business logic of software services graphically (see Fig. 3, right).

Note that both formats concentrate on standardizing the representation of relationships between business terms. Ideally, the standardization should also encompass the vocabulary (i.e., the terms themselves). For this purpose, we recommend the introduction and usage of controlled vocabularies. We did not make such vocabularies a part of the BoSDL since this would only limit its applicability, though. Instead, we present empirical indications in Sect. 5 that the BoSDL is able to improve the comparison of services even if the vocabulary slightly differs, for instance due to the use of synonyms.

4.3 Procedure

With the BoSDL, the business logic of software services can be described in three steps, which can be iterated as needed: (1) specification of the vocabulary, (2) specification of relationships, (3) creation of a mapping to programming interface elements. We illustrate the application of the BoSDL by specifying the business logic of a software service that fulfills all requirements given in Table 1. For reasons of compactness, we make use of the textual presentation format (see Table 4).

In the first step, service providers have to specify a vocabulary of business terms. To achieve this goal, relevant business terms have to be identified. The starting point for the identification of relevant business terms are the operations, which are accessible on the service interface (cf. line 38–45 of the interface specification depicted in Table 3). For each operation on the service interface that has a business-relevant meaning (note that this does not necessarily apply for auxiliary functions), business terms characterizing the functionality and the characteristic object of the operation have to be defined. Accordingly, the service provider specifies the term "define warehouse" to characterize the business logic of the operation `createStorage` on the service interface in our example scenario. To describe the characteristic object, the service provider specifies the term "warehouse", which has to be included into the vocabulary as well. Next to interface operations and characteristic objects, business terms also have to be defined to characterize parameters of operations and/or data types that possess a business-relevant meaning. Note that the provider can also define business terms to characterize additional functions and information objects, which have no representation at the service interface. Such a strategy is recommended to express further relevant aspects of the business logic. For instance, business terms for composite functions that aggregate interface operations can be defined. Additional business terms can also be defined to express aggregate information objects and/or attributes that are not visible on the programming interface. In our example scenario, the service provider for instance decides to also define the "commissioning strategy" as a relevant business term.

During the second step, statements are specified based on the vocabulary in order to express complex aspects of the implemented business logic. To achieve this task, the grammar of the presentation format has to be used. It allows specifying relationships between information objects, between information objects and functions, and between functions. In addition, the provider can define temporal courses of action between functions as processes. Using the sentence pattern for property declarations, the provider for instance states: "A warehouse has a commissioningStrategy". The form of the supported commissioning strategy could be specified by stating: "A commissioningStrategy is first-expires-first-out". This describes that only one commissioning strategy is supported and used throughout the warehouse.

In the third step, service providers ought to map business terms from the vocabulary to elements of the programming interface in order to depict, which elements of a software service realize certain parts of the specified business logic. In our example scenario, the term "warehouse" is mapped to the data type `Struc-`

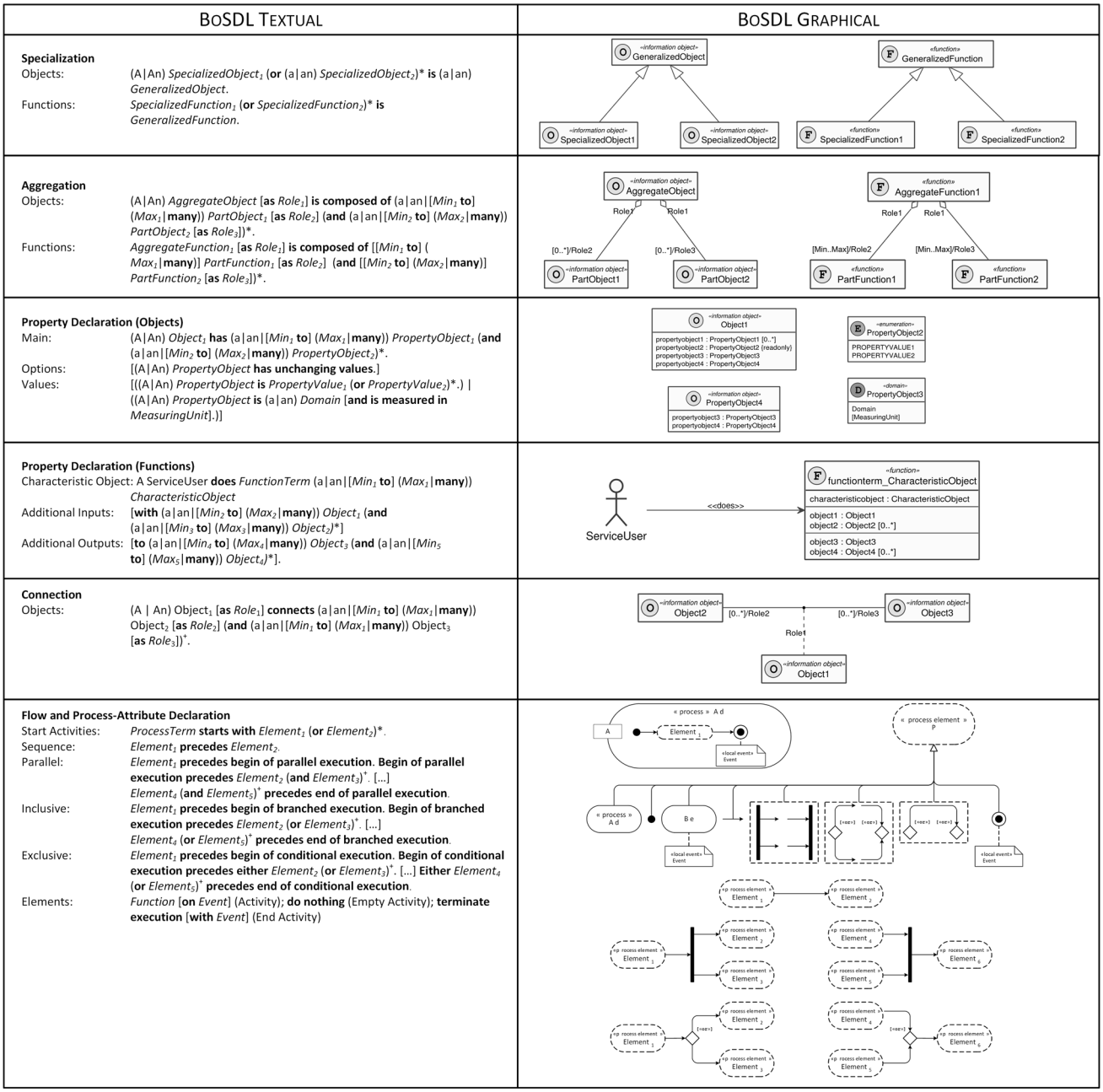

Fig. 3 Grammars to describe the business logic in textual (left) and graphical notation (right)

`tureStorage` to describe that it is realized by this interface element. For reasons of brevity, we shortened the description in Table 4. A more complete example specification is provided in the online appendix. The compilation of statements (step 2), however, is depicted in a sufficiently complete manner that allows verifying all functional requirements depicted in Table 1 as indicated by the numbering. As the content shows, the description nevertheless remains quite compact, aiming to keep the specification effort for service providers within acceptable limits. To verify, if a software service fulfills existing functional requirements, a service consumer has to assess if the tasks that are to be performed are supported adequately (i.e., with the required business logic). A manual assessment basically can be conducted in three steps: First, the service consumer has to verify that all tasks are actually implemented by operations on the service interface. Therefore, (s)he has to match the

Table 4 Exemplary statements to describe the business logic of a service fulfilling the requirements given in Table 1

I. Vocabulary	
Warehouse	A warehouse is a facility to stock articles
WarehouseArea	A warehouse area is a sector that is dedicated to carry out a specific warehousing task
CommissioningStrategy	A commissioningStrategy describes a method to retrieve goods from storage
Define warehouse	As an initial step, a warehouse administrator has to define the structure of the warehouse
Stockkeeper	A stockkeeper is tasked with maintaining warehouses
...	
II. Compilation of statements	
A warehouse **is composed of** 1 **to many** warehouseArea. A warehouse **has** a name and a commissioningStrategy. A commissioningStrategy **is** first-expires-first-out ✪. A warehouseArea **is composed of** 1 **to** 750,000 storageBin ⓪. A warehouseArea **has** an areaType. An areaType **is** storing **or** picking **or** shipping ②. A storageBin **has** a binType **and** a slot **and** a level ⑤ **and** 1 **to** 10 articleCounts ⑦ **and** a storageStrategy. A storageStrategy **is** static or chaotic ③. A binType **has** a height **and** a width **and** a depth **and** a tonnageCapacity ④ **and** a storageUnit. A storageUnit **is** standardPallet **or** singleArticle ⑥. An articleCount **has** an articleID **and** a count. A count **is** an Integer. A height **is** a RealNumber. A slot **is** a CharacterSequence.	
A stockkeeper **does** manage 0 **to** 5 warehouse ①. A stockkeeper **does** create a commissioningPlan **with** a customerOrder. A stockkeeper **does** create a storagePlan **with** a deliveryReceipt ⑨. A stockkeeper **does** book an inventoryChange **with** a reservation ⑩. An administrator **does** define a warehouse ⑧.	
III. Interface mapping	
Warehouse	StructureStorage
WarehouseArea	StructureStorageArea
Define warehouse	CreateStorage
...	

tasks contained in the requirements description against the functions described in the BoSDL document. In our example scenario, a service consumer for instance has to evaluate if the service provides a function to create, read, update, and delete warehouse structures ⑧. The assessment is successful if all tasks can be mapped onto functions with appropriate business logic. In a second step, the service consumer has to verify, if the entities, which are to be processed, are supported adequately. Therefore, (s)he has to match the entities contained in the requirements description against the information objects described in the BoSDL document. In the example scenario, the service consumer for instance has to evaluate if the service is able to manage warehouses with up to 5000 storage cells ⓪. The assessment is successful, if all entities can be mapped onto information objects with appropriate business logic. In a third step, the service consumer ought to verify that the tasks can be executed in the intended temporal order using the service. Therefore, (s)he has to inspect if possible temporal courses of action, which are specified as processes in the BoSDL document, conform to the manner in which the tasks shall be executed. Note that we are unable to depict a detailed requirements matching process due to space limitations. Instead, we refer to the literature (e.g., Kluge et al. 2008; Platenius et al. 2017).

5 Evaluation

During our DSR endeavor, we conducted various ex-post evaluations of the BoSDL as recommended by Sonnenberg and vom Brocke (2012) and Venable et al. (2016). To proof the applicability of the BoSDL, we used it to describe the business logic of a complex software service supporting the booking of tickets in the aviation domain and that of several services supporting the inventory management. In both cases, we were able to express all relevant content with the final language version. To proof the technical feasibility, we developed a modeling tool that is able to convert specifications between the two language formats. In addition, we implemented compilers that verify specifications against the language grammars (see online appendix) and integrated support for the BoSDL into a prototypical service marketplace.

5.1 Proof of Requirements

Before subjecting the BoSDL to detailed evaluations, we examined if its design fulfills requirements R1–R6 (Sect. 2.3). Ideally, a solution should cover all concepts necessary to specify the business logic of the operations available on the service interface (R1). While we are unable to prove completeness, we verified that the designed approach is able to specify the business logic of software services across various domains by applying it to complex use cases as described above. During the iterations of our

study, we furthermore observed a stabilization effect: While we were not able to express all relevant content adequately and had to refine the language concept during the first three iterations, we did not observe such problems later on (Sect. 3). Although we cannot guarantee completeness, we would hence argue that the BoSDL sufficiently meets R1.

An ideal solution furthermore needs to provide non-redundant, unambiguous constructs (R2). During the design of the BoSDL, we only added language concepts as long as they expressed semantically different parts of the business logic. To also ensure that each part of the business logic can only be expressed by one language construct, we carefully analyzed service descriptions that we created as described above. As we could identify no ambiguities, we conclude that the BoSDL fulfills R2.

By means of its meta-model, the BoSDL prescribes both the types of concepts and the types of relationships that can be used to describe the business logic of software services. The grammars of the presentation formats moreover prescribe the format of service descriptions. The BoSDL hence establishes normative rules for the description of the business logic as demanded by R3.

As a core element of the BoSDL, we have deliberately designed a presentation format that expresses the business logic using (a standardized form of) natural language. We deem this format to be straightforwardly understandable both for business experts and software designers, which might be involved in the assessment and selection of software services. For designers, we also provide a graphical format that is based on the UML as a de-facto standard. The BoSDL thus provides an understandable presentation format as required by R4.

To be efficiently applicable, an ideal language furthermore needs to introduce a compact set of constructs (R5). During the design of the BoSDL, we only introduced concepts that were found necessary to describe the business logic of the operations available of a service. The BoSDL distinguishes three concept types and eight relationship types. In contrast to general-purpose modeling languages such as ARIS or MEMO, it moreover depicts the content using a single presentation format. Compared to such approaches, we would therefore argue that the BoSDL satisfies R5. To confirm that the language is indeed perceived as lightweight, it would be necessary to evaluate the language in use, however.

In comparison, the fulfillment of R6 is obvious. The BoSDL is able to express the business logic of software services independently of their technological realization, because it is described in form of a conceptual model, which is technology-agnostic by definition.

5.2 Proof of Usefulness

To proof the usefulness, we decided to examine if the BoSDL expedites the assessment and selection of software services as its intended usage context. In particular, we wanted to know if it enhances the ability of consumers to assess the business logic of software services and to identify the ones best fulfilling their functional requirements. As we wanted to make a causal inference, namely that the description language is responsible for the observed effects, with a high degree of internal validity, we decided to control potentially confounding influences to the best possible extent. In so doing, we followed recommendations to maximize internal validity in order to achieve a rigorous testing of the artifact (Calder et al. 1982). We therefore conducted a laboratory experiment, in which software services had to be assessed by participants based on varying service descriptions. In the following, we adopt the structure for the presentation of experimental research proposed by Wohlin et al. (2012) to describe the design of our laboratory experiment.

5.2.1 Experiment Design

5.2.1.1 Goals, Hypotheses and Variables The purpose of the evaluation was to statistically test propositions P1–P3 (Sect. 2.3), that is, to investigate if the BoSDL allows consumers to identify eligible software services with a higher degree of effectiveness, efficiency, and satisfaction. The main observed factor was the service description approach. To vary the description approach, we randomly assigned participants to three groups. The control group received a general-purpose description of the services and a specification of the interface. The provided information corresponds to what is typically made available to consumers on today's SaaS marketplaces. The interface was described using WSDL and annotated with business terms using the WSDL documentation feature (see Table 3). Following Vitharana et al. (2003), we annotated each function and parameter with a business term. In so doing, we were able to evaluate the BoSDL against the current state of the art in research and practice (Sect. 2.1). The treatment groups additionally received a description of the business logic, denoted either in the textual or graphical BoSDL format. To account for individual characteristics, we included control variables such as age, gender, or prior experience.

5.2.1.2 Procedure To test the propositions, we investigated a realistic service selection scenario based on the case discussed in Sect. 2.2. The required service descriptions were derived from our endeavor to describe inventory management services (Sect. 5). The experiment focuses on

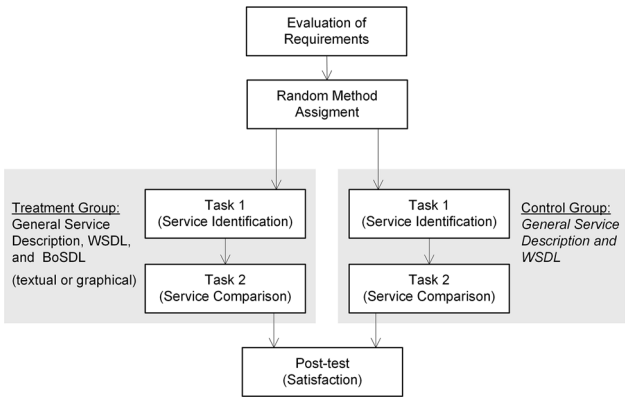

Fig. 4 Experiment procedure

the assessment of software services against functional requirements, which is one task in such a scenario (Kontio 1996). To fulfill this task, consumers typically engage in two activities that we incorporated into the experiment (see Fig. 4): First, the participants had to match service descriptions against a set of functional requirements and to assess if the services fulfill all of them. Second, they had to compare the business logic of two services and to name any differences.

The experiment began with an assessment of prior knowledge and experience. Next, the participants received the requirements document and were given time to study it. The requirements document had a length of 322 words and a content comparable to the one depicted in Table 1. Thereafter, the participants were given access to the service descriptions depending on the group they were assigned to. During the first task, the participants had to compare six service candidates and to identify those fulfilling all functional requirements. The service candidates all had a comparable complexity (e.g., the WSDL files varied between 1342 and 1501 lines of code) but varied in their support of the requirements and in the used vocabulary (e.g., "storage" or "depot" instead of "warehouse"). Varying the business terms allowed us to examine if the BoSDL supports selecting software services even in the likely scenario that providers use different business terms for the same concepts. Moreover, we varied the business terms so that none of them would exactly match the ones used in the requirements specification. Of the six service candidates, only one fulfilled all requirements. The others varied in their support of requirements ①–⑦ or did not provide all required operations (⑨ and ⑩, see Table 1). During the second task, the participants had to compare the descriptions of two service candidates, whose business logic varied in four aspects: the maximum of manageable warehouses, a customizable storage strategy for warehouse areas, the limit of articles per storage cell, and the support of a static storage strategy. After each task, the participants had to specify their results on a standardized form sheet. In addition, they had to indicate if the provided service description was sufficient to come to a conclusion. The experiment ended with a short survey, in which we evaluated the satisfaction of the participants with their service description language. The survey consisted of six questions (Table 5). There was no time restriction for completing the tasks. To measure efficiency, we recorded the task completion time of each participant, though.

5.2.1.3 Subjects Following our intent to measure the effect of the developed description approach on the ability to select software services and to control confounding effects to the best possible extent, we decided to chose students instead of practitioners as surrogates of consumers. In a setting such as ours, students are often recommended as participants for several reasons (Anderson 1982; Vessey and Conger 1994; Gemino and Wand 2004): First, experienced practitioners typically build upon their prior knowledge of previously applied service selection procedures, which can superimpose the effects of the approach that is to be examined. In contrast, students do not possess a comparably large and potentially varying store of alternative methods. Moreover, practitioners have usually automated their problem-solving processes to a certain extent already. They might hence be less willing or able to adapt to new processes than students and novice users. For this reason, it might also be easier to teach novice users to apply a specific approach than it is to teach people who may already be experts with other methodologies.

While the chosen setting allowed us to examine if the designed approach basically is useful, we will also have to test its effect in a real-world scenario with practitioners in order to strengthen the external validity of our findings. Literature furthermore provides indications that expert users with profound knowledge might even gain in efficiency when using language dialects that, for instance, implement more complex constructs and/or a reduced symbol set (i.e., a "pro version" of the language, Kalyuga et al. (2003). As we could not study such effects with the chosen evaluation strategy, the obtained results remain limited in this respect as well.

To approximate a natural setting, we invited senior students from a relevant field of study. In total, 126 students from computer science (30), information systems (74), and business administration (22) degree programs voluntarily participated in the experiment. 103 were male and 23 female. On average, they were in their fifth semester. Each participant had computer science as major and attended a lecture course in component & service engineering, in which service selection procedures were taught. Apart from the contents taught in class, four participants

Table 5 Post-test questions

	Question	Scale
Q1	Do you think the service description contains all necessary information?	(Not at all) 1 2 3 4 5 6 (completely)
Q2	Do you think the service description was concise?	(Not at all) 1 2 3 4 5 6 (completely)
Q3	Do you think the service description was clearly structured?	(Not at all) 1 2 3 4 5 6 (completely)
Q4	Do you think the service description was easy to use?	(Not at all) 1 2 3 4 5 6 (completely)
Q5	Would you use the service description language again?	(No) 0 1 (yes)
Q6	Would you recommend the service description language?	(No) 0 1 (yes)

stated to have prior knowledge about service selection procedures.

5.2.1.4 Data Collection and Analysis Procedure To measure effectiveness, we determined the accuracy with which the participants reached their goals (Frøkjær et al. 2000). For task one, we granted a score of 1.0 for the correct answer and discounted 0.5 for incorrect answers. Doing so allowed us to only count assessment results that had more appropriate than unsuitable answers. Efficiency generally is defined as relation between accuracy and invested effort (Frøkjær et al. 2000). Accordingly, it was measured as ratio between the effectiveness score and the task completion time. It can be interpreted as points per minute score. Satisfaction was measured using the scales depicted in Table 5. We applied ordinal logistic regressions to analyze the scores, since the dependent variables mostly have an ordinal scale and the independent variables have discrete values. Ordinal logistic regression is regarded as standard method for such scenarios (Kutner et al. 2005). Although our propositions suggested one-tailed testing, we conducted two-tailed tests since they provide stricter results.

5.2.2 Experiment Results

As shown in Table 6, the treatment groups achieved significantly higher effectiveness scores than the control group in both tasks. The results hence support P1. In particular, they confirm that the BoSDL increases the ability of consumers to identify software services, which match functional requirements (task 1). In this respect, the results between the textual and the graphical BoSDL format groups were almost identical. The results furthermore show that the BoSDL increases the ability of consumers to identify differences in the business logic of software services (task 2). In task 2, the group working with the textual BoSDL format even achieved slightly better results than users of the graphical BoSDL format. Table 6 also shows that the treatment groups achieved significantly higher efficiency scores than the control group in both tasks. The results hence also support P2. Among the BoSDL formats, the results again are almost identical with respect to task 1. In task 2, users of the textual format were slightly more efficient.

With respect to the satisfaction (P3), we obtained mixed results. The question, if the description contains all necessary information (Q1), returned no significant differences between the groups. The results for Q2–Q4 were in line with our assumption: Both BoSDL groups perceived their formats to be more clearly structured, more concise, and easier to use than the control group. Regarding Q5 and Q6, the results vary largely. While significantly more users of the graphical BoSDL format would reuse and recommend their language compared to the control group, users of the textual BoSDL format showed no such intents although they achieved better results. Accordingly, P3 has not been fully confirmed.

To ensure the robustness of our tests and examine if the variations are indeed explained by the treatment, we included the above-mentioned control variables into our regression models. We found gender to have a significant influence on the accuracy in task 1. Since our treatment variable nevertheless showed significant results, however, this was not deemed to be a problem. As for the other models, the control variables had no significant effects. Moreover, the likelihood ratio chi-square values (all > 25) demonstrate that all our models are statistically significant compared to the models without predictors.

5.2.3 Interpretation

The results indicate that the BoSDL helps consumers to more effectively select service candidates, which fulfill functional requirements, and assess differences in the business logic between services. It also allows them to more efficiently accomplish such tasks. The observed effect varies only little between the two representation formats of the BoSDL. It seems hence largely to be caused by the language concept, that is, the provided content. In contrast, general-purpose descriptions and specifications of the programming interface clearly did not support participants equally well during the assessment, although they were even annotated with business terms as suggested by

Table 6 Test results and summary statistics

	Ordinal logistic regression results		Treatment > control?	Summary statistics					
	Coefficient	*p* value			Min	Max	Mean	Median	Std. dev.
Effectiveness									
Tl: Accuracy	BoSDL textual: $\beta = 1.330$	0.003**	✓	Control	0.00	1.00	0.23	0.00	0.39
	BoSDL graphical: $\beta = 1.332$	0.003**	✓	BoSDL-T	0.00	1.00	0.51	0.50	0.45
				BoSDL-G	0.00	1.00	0.51	0.50	0.46
T2: Accuracy	BoSDL textual: $\beta = 3.203$	0.000***	✓	Control	0.00	3.00	1.11	1.00	0.87
	BoSDL graphical: $\beta = 2.195$	0.000***	✓	BoSDL-T	1.00	4.00	3.02	3.00	0.96
				BoSDL-G	0.00	4.00	2.42	2.50	1.15
Efficiency									
Tl: Accuracy/time	BoSDL textual: $\beta = 1.350$	0.002**	✓	Control	0.00	0.04	0.01	0.00	0.01
	BoSDL graphical: $\beta = 1.350$	0.002**	✓	BoSDL-T	0.00	0.05	0.02	0.01	0.02
				BoSDL-G	0.00	0.05	0.02	0.01	0.01
T2: Accuracy/time	BoSDL textual: $\beta = 2.965$	0.000***	✓	Control	0.00	0.22	0.07	0.07	0.06
	BoSDL graphical: $\beta = 2.238$	0.000***	✓	BoSDL-T	0.07	0.60	0.23	0.20	0.11
				BoSDL-G	0.00	0.58	0.18	0.18	0.11
Satisfaction									
Ql: Information	BoSDL textual: $\beta = 0.375$	0.348	✓	Control	0.00	6.00	4.07	5.00	1.52
	BoSDL graphical: $\beta = 0.640$	0.108	✓	BoSDL-T	2.00	6.00	4.43	5.00	1.27
				BoSDL-G	0.00	6.00	4.55	5.00	1.37
Q2: Conciseness	BoSDL textual: $\beta = 0.871$	0.029*	✓	Control	0.00	6.00	3.38	3.50	1.56
	BoSDL graphical: $\beta = 0.787$	0.043*	✓	BoSDL-T	1.00	6.00	4.12	4.00	1.43
				BoSDL-G	0.00	6.00	4.02	4.00	1.42
Q3: Clarity	BoSDL textual: $\beta = 1.094$	0.007**	✓	Control	1.00	6.00	2.52	2.00	1.63
	BoSDL graphical: $\beta = 1.384$	0.001***	✓	BoSDL-T	1.00	6.00	3.43	4.00	1.55
				BoSDL-G	0.00	6.00	3.69	4.00	1.49
Q4: Easy to use	BoSDL textual: $\beta = 1.932$	0.000***	✓	Control	1.00	6.00	3.10	3.00	1.34
	BoSDL graphical: $\beta = 1.525$	0.000***	✓	BoSDL-T	1.00	6.00	4.48	4.50	1.33
				BoSDL-G	0.00	6.00	4.14	4.00	1.30
Q5: Reuse	BoSDL textual: $\beta = 0.479$	0.276	✓	Control	0.00	1.00	0.45	0.00	0.50
	BoSDL graphical: $\beta = 2.193$	0.000***	✓	BoSDL-T	0.00	1.00	0.57	1.00	0.50
				BoSDL-G	0.00	1.00	0.88	1.00	0.33
Q6: Recommend	BoSDL textual: $\beta = 0.397$	0.375	✓	Control	0.00	1.00	0.36	0.00	0.48
	BoSDL graphical: $\beta = 1.751$	0.000***	✓	BoSDL-T	0.00	1.00	0.45	0.00	0.50
				BoSDL-G	0.00	1.00	0.76	1.00	0.43

T task, *Q* question

***0.1% (2-tailed) significance, **1% (2-tailed) significance, *5% (2-tailed) significance

Vitharana et al. (2003). We can hence conclude that the BoSDL advances the current state of the art in functional service description.

Regarding the perceived clarity, conciseness, and ease of use, the observed results lend additional support to our claims that the BoSDL fulfills R2 (clarity), R3 (strictness), and R5 (simplicity). During the experiment, we could furthermore observe that most participants of the treatment groups did not look at the provided WSDL files at all. This observation lends support to the assumption that the BoSDL provides a sufficiently complete solution (R1). Interestingly, however, the participants of all groups ranked the provided information equally to be complete. The reasons for this result seem to be two-fold: On the one hand, we decided to only formulate requirements that could be evaluated using the WSDL specification. The control group hence also found the provided information to be generally complete. On the other hand, this perception apparently also was provoked by the sheer size of the WSDL files.

With almost 90% of the participants stating that they would reuse the graphical format and 75% noting that they would recommend it to others, we can furthermore assume that there also exists a potential for acceptance of the BoSDL in practice. Clearly, however, we will have to examine further why these results could not be observed for the textual format as well.

6 Implications and Conclusions

The fulfillment of existing functional requirements typically is the most important criterion when assessing and selecting software services from the cloud (Repschlaeger et al. 2012; Polyviou et al. 2014). Yet, approaches to document and evaluate the business logic of software services – i.e., the provided functionality from a business-oriented perspective – are still rare (Sun et al. 2014). To contribute to the closure of this literature gap, we proposed the BoSDL as an approach to specify the business logic of software services by means of a conceptual model. In contrast to general-purpose conceptual modeling approaches such as ARIS or MEMO, the BoSDL aims at creating smaller models that are tailor-made to document the operations available on the service interface. Furthermore, it uses only a single notation to depict the conceptual model. We consider such a lightweight approach to service description to be essential to keep the effort, which is required to use the language, within reasonable limits and facilitate its acceptance in practice (O'Sullivan 2006). Other than current functional service description approaches, the BoSDL particularly supports specifying complex aspects of the business logic by interrelating the elements of the conceptual model. The results of our evaluation indicate that this approach can significantly enhance the ability of consumers to assess software services against their functional requirements.

The results of our DSR study have implications for academia and practice alike. As regards practice, we deliver a lightweight approach that allows providers to make the business logic of their software services explicit. Seen from a consumer's perspective, the information provided with such an approach has a significant potential to expedite the assessment and selection of software services. While this potential particularly becomes relevant when composing modular SaaS solutions from multiple services, a description of the business logic can also facilitate the selection of singular COTS software packages (or software services). Especially if their business logic is complex and has to be evaluated on the basis of test versions, a lack of attention to functional requirements is known to be a practical problem in selecting COTS software packages, too (Kontio 1996). For providers, specifying the business logic of their services can create a competitive advantage, since it becomes easier and more promising for consumers to take such services into consideration. As the BoSDL builds upon conceptual models, providers will probably be able to reuse results from the conceptual design phase, thus minimizing the required effort. We agree, however, that we will have to examine the incentives and efforts for providers more closely in future research iterations. Based on the results of further evaluations and applications of the BoSDL, we also plan to formulate best practices that show how the language can be most efficiently used to describe the business logic of services and map it to functional requirements.

As regards academia, the results of our study contribute to the emerging SSE discipline, which "seeks to advance knowledge on models, methods, and artifacts that enable or support the engineering of service systems" (Böhmann et al. 2014). In particular, we provide evidence-based knowledge that advances the design of service description methods. In the SSE discipline, service descriptions have repeatedly been emphasized as an important measure to help matching the provider's offering to the consumer's needs (Maglio et al. 2009; Ferrario et al. 2012), thereby particularly supporting the design and configuration of modular SaaS systems (Turner et al. 2003; Repschlaeger et al. 2012; Ferrario et al. 2012). By showing how conceptual modeling theory can be used to describe the business logic of software services in detail, we complement existing service description approaches in the SaaS domain, which so far rather focus on describing the quality, the programming interface, and/or contractual properties of software services (Polyviou et al. 2014; Sun et al. 2014). While such approaches assume "that the functions of the evaluated services are by default matched to users' requirements" (Sun et al. 2014), the presented BoSDL specifically focuses on supporting this task. Following our research goal (Sect. 3), the central knowledge contribution of our DSR study is a nascent design theory with operational principles (Gregor and Hevner 2013) regarding the description of the business logic. While we designed the BoSDL with software services in mind, the language concept is independent of the technical realization. Although we have not examined the generalizability of the BoSDL yet, it hence might also be usable to describe the business logic of other kinds of services.

There also exist limitations in the light of which the study results have to be interpreted. Most prominently, the external validity of our findings is still limited because we have mainly applied the BoSDL in selected use cases and controlled experiments until now. Although the use cases were taken from different application domains and complex in nature, we need to further examine the suitability of the BoSDL in practical scenarios, for instance by

conducting field studies. So far, we furthermore used students as surrogates of service consumers in our experiments (Sect. 5.2.1). While the results indicate that the BoSDL can even support novice users during the assessment and selection of software services, we will have to confirm our observations in naturalistic settings with experienced practitioners. Generally, a notation that is efficient for novice users does not necessarily have to be ideal for experienced experts, since they might be able to use a denser language with more complex constructs or fewer symbols even more efficiently (Kalyuga et al. 2003).

Besides efforts to further strengthen the validity of our findings, the outcomes of our DSR endeavor provide several additional avenues for future research. A promising measure to further facilitate the selection of services would be to (partially) automate the assessment against existing functional requirements. However, in the absence of domain standards – which would provide ideal support for such endeavors – it would be necessary to deal with heterogeneous business terms and domain conceptions. To provide better support for automated use in heterogeneous scenarios, the meta-model of the BoSDL would have to be extended with features to support ontology management (e.g., a thesaurus) and automated reasoning with inconsistent ontologies. Future research should also investigate the potential of new presentation formats. While the objective performance of the users did not depend on the presentation format during our laboratory experiment, the subjective perception of the two formats varied considerably. It is hence conceivable that there might exist more intuitive formats to depict the business logic of services, especially for different types of users. With the results presented in this paper, we hope to provide a starting point for such endeavors.

References

Akkiraju R, Farrell J, Miller J, Nagarajan M, Schmidt M-T, Sheth A, Verma K (2005) Web service semantics – WSDL-S, version 1.0. Technical report, World Wide Web Consortium

Anderson JR (1982) Acquisition of cognitive skill. Psychol Rev 89(4):369–406

Atkinson B, Della-Libera G, Hada S, Hondo M, Hallam-Baker P, Klein J, LaMacchia B, Leach P, Manferdelli J, Maruyama H, Nadalin A, Nagaratnam N, Prafullchandra H, Shewchuk J, Simon D (2002) Web services security (WS-Security) version 1.0. Technical report, World Wide Web Consortium

Barros A, Oberle D (eds) (2012) Handbook on service description – USDL and its methods. Springer, Heidelberg

Bodart F, Patel A, Sim M, Weber R (2001) Should optional properties be used in conceptual modelling: a theory and three empirical tests. Inf Syst Res 12(4):384–405

Böhmann T, Leimeister JM, Möslein K (2014) Service systems engineering: a field for future information systems research. Bus Inf Syst Eng 6(2):73–79

Bunge M (1977) Treatise on basic philosophy: ontology I. The furniture of the world, vol 3. Reidel, Boston

Cabrera F, Copeland G, Cox B, Freund T, Klein J, Storey T, Thatte S (2002) Web services transaction (WS-Transaction) version 1.0. Technical report, World Wide Web Consortium

Calder BJ, Phillips LW, Tybout AM (1982) The concept of external validity. J Consum Res 9(3):240–244

Chinnici R, Moreau J-J, Ryman A, Weerawarana S (2007) Web services description language (WSDL) version 2.0 part 1: core language. Technical report, World Wide Web Consortium

Davis R, Shrobe HE, Szolovits P (1993) What is a knowledge representation? AI Mag 14(1):17–33

Eisa M, Younas M, Basu K, Zhu H (2016) Trends and directions in cloud service selection. In: IEEE symposium on service-oriented system engineering, IEEE, pp 423–432

Ferrario R, Guarino N, Trampus R, Laskey K, Hartman A, Gangadharan GR (2012) Service system approaches – conceptual modeling approaches for services science. In: Barros A, Oberle D (eds) Handbook of service description: USDL and its methods. Springer, Heidelberg, pp 75–109

Frank U (2013) Domain-specific modeling languages: requirements analysis and design guidelines. In: Reinhartz-Berger I, Sturm A, Clark T, Cohen S, Bettin J (eds) Domain engineering: product lines, languages, and conceptual models. Springer, Heidelberg, pp 133–157

Frank U (2014) Multi-perspective enterprise modeling: foundational concepts, prospects and future research challenges. Softw Syst Model 13(3):941–962

Frøkjær E, Hertzum M, Hornbæk K (2000) Measuring usability: Are effectiveness, efficiency, and satisfaction really correlated. In: SIGCHI conference on human factors in computing systems. ACM, New York, NY, USA, pp 345–352

Gehani N, McGettrick AT (1986) Software specification techniques. Addison-Wesley, Wokingham

Gemino A, Wand Y (2004) A framework for empirical evaluation of conceptual modeling techniques. Requir Eng 9(3):248–260

Goldkuhl G, Lind M, Seigerroth U (1998) Method integration: the need for a learning perspective. IEE Proc Softw 145(4):113–118

Gómez-Pérez A, Fernández-López M, Corcho O (2004) Ontological engineering. Springer, Heidelberg

Goodman N (1976) Languages of art: an approach to a theory of symbols. Hackett, London

Graham S, Hull D, Murray B (2006) Web services base notification 1.3. Technical report, OASIS Standard

Gregor S, Hevner AR (2013) Positioning and presenting design science research for maximum impact. MIS Q 37(2):337–355

Hadar I, Soffer P (2006) Variations in conceptual modeling: classification and ontological analysis. J Assoc Inf Syst 7(8):569–593

Herbert L, Parks S, O'Donnell G, Erickson J, Caputo M, Nagel B (2016) The ROI of software as a service. Technical report, Forrester

Hevner AR, March ST, Park J, Ram S (2004) Design science in information systems research. MIS Q 28(1):75–105

Hrach C, Alt R (2018) Functional service description on service marketplaces. In: Multikonferenz Wirtschaftsinformatik

IBM (2014) Champions of software as a service: How SaaS is fueling powerful competitive advantage. Technical report, IBM Corporation

Iivari J (2007) A paradigmatic analysis of information systems as a design science. Scand J Inf Syst 19(2):39–63

Iwasa K (2004) Web services reliable messaging. Technical report, OASIS Standard

Kalyuga S, Ayres P, Chang SK, Sweller J (2003) The expertise reversal effect. Educ Psychol 38(1):23–31

Kluge R, Hering T, Belter R, Franczyk B (2008) An approach for matching functional business requirements to standard application software packages via ontology. In: 32nd annual IEEE international computer software and applications conference, pp 1017–1022

Kontio J (1996) A case study in applying a systematic method for COTS selection. In: 18th international conference on software engineering, pp 201–209

Kutner MH, Nachtsheim CJ, Neter J, Li W (2005) Applied linear statistical models, 5th edn. The McGraw-Hill/Irwin series: operations and decision sciences. McGraw-Hill, Boston

Land R, Blankers L, Chaudron MRV, Crnkovic I (2008) COTS selection best practices in literature and in industry. In: 10th international conference on software reuse, pp 100–111

Le L-S, Ghose A, Morrison E (2010) Definition of a description language for business service decomposition. In: 1st international conference on exploring services sciences

Ludwig H, Keller A, Dan A, King RP, Franck R (2003) Web service level agreement (WSLA) language specification. Technical report, IBM Corporation

Maglio PP, Vargo SL, Caswell N, Spohrer J (2009) The service system is the basic abstraction of service science. Inf Syst e-Bus Manag 7(4):395–406

Martin D, Paolucci M, McIlraith S, Burstein M, McDermott D, McGuinness D, Parsia B, Payne T, Sabou M, Solanki M, Srinivasan N, Sycara K (2005) Bringing semantics to web services: the OWL-S approach. In: Cardoso J, Sheth AP (eds) Semantic web services and web process composition, vol 3387, San Diego, CA, USA. Springer, Heidelberg, pp 26–42

Moody DL (2009) The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. IEEE Trans Softw Eng 35(6):756–779

Mylopoulos J (1992) Conceptual modeling and telos. In: Loucopulos P, Zicari R (eds) Conceptual modeling, databases, and CASE: an integrated view of information systems development. Wiley, Cambridge, pp 49–68

Oaks P, ter Hofstede AHM, Edmond D (2003) Capabilities: describing what services can do. In: First international conference on service-oriented computing, pp 1–16

Ortner E, Schienmann B (1996) Normative language approach – a framework for understanding. In: 15th international conference on conceptual modeling, pp 261–276

O'Sullivan J (2006) Towards a precise understanding of service properties. PhD thesis, Queensland University of Technology

Pautasso C, Zimmermann O, Amundsen M, Lewis J, Josuttis NM (2017) Microservices in practice, part 1: reality check and service design. IEEE Softw 34(1):91–98

Peffers K, Tuunanen T, Rothenberger MA, Chatterjee S (2007) A design science research methodology for information systems research. J Manag Inf Syst 24(3):45–77

Platenius MC, Shaker A, Becker M, Huellermeier E, Schaefer W (2017) Imprecise matching of requirements specifications for software services using fuzzy logic. IEEE Trans Softw Eng 43(8):739–759

Pohl K (2010) Requirements engineering: fundamentals, principles, and techniques. Springer, Heidelberg

Polyviou A, Pouloudi N, Rizou S (2014) Which factors affect software-as-a-service selection the most? A study from the customers and the vendors perspective. In: 47th Hawaii international conference on system sciences, pp 5059–5068

Repschlaeger J, Wind S, Zarnekow R, Turowski K (2012) Selection criteria for software as a service: an explorative analysis of provider requirements. In: Americas conference on information systems, Seattle

Roman D, Keller U, Lausen H, de Bruijn J, Lara R, Stollberg M, Polleres A, Feier C, Bussler C, Fensel D (2005) Web service modeling ontology. Appl Ontol 1(1):77–106

Rowsell-Jones A, Lowendahl J-M, Howard C, Nielsen T (2016) The 2017 CIO agenda: seize the digital ecosystem opportunity. Technical report, Gartner Inc

Scheer AW (2000) ARIS – business process frameworks, 3rd edn. Springer, Heidelberg

Schlauderer S, Overhage S (2011) How perfect are markets for software services? An economic perspective on market deficiencies and desirable market features. In: European conference on information systems

Shanks G, Tansley E, Weber R (2003) Using ontology to validate conceptual models. Commun ACM 46(10):85–89

Sonnenberg C, vom Brocke J (2012) Evaluations in the science of the artificial-reconsidering the build-evaluate pattern in design science research. In: Information systems, advances in theory and practice

Sun L, Dong H, Hussain OK, Hussain FK, Chang E (2014) Cloud service selection: state-of-the-art and future research directions. J Netw Comput Appl 45:134–150

Terzidis O, Oberle D, Friesen A, Janiesch C, Barros A (2012) The internet of services and usdl. In: Barros A, Oberle D (eds) Handbook of service description: USDL and its methods. Springer, Heidelberg, pp 1–16

Topi H, Ramesh V (2002) Human factors research on data modeling: a review of prior research, an extended framework and future research directions. J Database Manag 13(2):3–15

Turner M, Budgen D, Brereton P (2003) Turning software into a service. IEEE Comput 36(10):38–44

UDDI Organization (2002) UDDI open draft specification. Technical report, UDDI Organization

Van der Aalst WMP, ter Hofstede AHM, Kiepuszewski B, Barros AP (2003) Workflow patterns. Distrib Parallel Databases 14(1):5–51

Vargo SL, Lusch RF (2004) Evolving to a new dominant logic for marketing. J Mark 68(1):1–17

Venable J, Pries-Heje J, Baskerville R (2016) Feds: a framework for evaluation in design science research. Eur J Inf Syst 25(1):77–89

Vessey I, Conger SA (1994) Requirements specification: learning object, process, and data methodologies. Commun ACM 37(5):102–113

Vitharana P, Zahedi F, Jain H (2003) Knowledge-based repository scheme for storing and retrieving business components: a theoretical design and an empirical analysis. IEEE Trans Softw Eng 29(7):649–664

Vukovic M, Laredo J, Muthusamy V, Slominski A, Vaculin R, Tan W, Naik V, Silva-Lepe I, Kumar A, Srivastava B, Branch JW (2016) Riding and thriving on the API hype cycle. Commun ACM 59(3):35–37

Wand Y, Weber R (1993) On the ontological expressiveness of information systems analysis and design grammars. J Inf Syst 3(4):217–237

Wang X, Vitvar T, Kerrigan M, Toma I (2006) A QoS-aware selection model for semantic web services. In: Service-oriented computing ICSOC 2006, pp 390–401

Webster J, Watson RT (2002) Analyzing the past to prepare for the future: writing a literature review. MIS Q 26(2):xiii–xxiii

Weyuker EJ (2001) The trouble with testing components. In: Councill WT, Heineman GT (eds) Component-based software engineering: putting the pieces together. Addison-Wesley, Upper Saddle River, NJ, pp 499–512

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer, Heidelberg

Zowghi D, Coulin C (2005) Requirements elicitation: a survey of techniques, approaches, and tools. In: Engineering and managing software requirements, Springer, Heidelberg, pp 19–46