2010

WHAT'S IN A SERVICE? SPECIFYING THE BUSINESS SEMANTICS OF SOFTWARE SERVICES

Sven Overhage
University of Augsburg, sven.overhage@wiwi.uni-augsburg.de

Sebastian Schlauderer
University of Augsburg, sebastian.schlauderer@uni-bamberg.de

WHAT'S IN A SERVICE? SPECIFYING THE BUSINESS SEMANTICS OF SOFTWARE SERVICES

Completed Research Paper

Sven Overhage and Sebastian Schlauderer
Component and Service Engineering Research Group
University of Augsburg, Germany
{sven.overhage, sebastian.schlauderer}@wiwi.uni-augsburg.de

Abstract

The success of the service-oriented computing (SOC) paradigm considerably depends on the ability of service consumers to distinguish between published services and choose the ones best suited for a development project. Current SOC standards primarily give information about technical service properties such as the programming interface and the binding information. This enables designers to analyze the technical compatibility of services with the rest of the system. On the basis of such technical information, it is difficult to assess which business semantics a service actually implements and whether it is suited to satisfy functional requirements, however. In this paper, we therefore propose the WS-Functionality language which allows providers to specify the business semantics of software services in business terms. In a design science approach, we firstly describe how conceptual models, which contain business terms and relationships between them, can be used to specify the business semantics of services. Building upon this solution concept, we present the language constructs of WS-Functionality and show a prototypic implementation as proof-of-concept. In a controlled experiment, we were able to support our claim that the information provided with WS-Functionality enhances the ability of service consumers to analyze the business semantics of services and judge whether it satisfies existing functional requirements.

Keywords: Service-oriented computing, software services, business semantics, conceptual model

Introduction

With its modular concept of reusing existing software services to compose business applications, service-oriented computing (SOC) promises a whole range of advantages. By allowing developers to break down an application building task into a set of smaller services, SOC introduces a well-proven strategy for complexity management that is known as divide-and-conquer and widely used in other engineering disciplines (Kanigel 1997; Speed et al. 2001). The distributed, loose coupling of services from different providers facilitates adapting or replacing application parts that are affected by changes in the business environment (Erl 2005; McGovern et al. 2006). Reusing services from in-house and external providers finally contributes to reducing the development time and delivering applications faster (McGovern et al. 2006; Papazoglou et al. 2007).

As a consequence, SOC and especially its realization based on the Web services technology stack have attracted great attention in research and industry (Weerawarana et al. 2005). Today, SOA has been widely accepted as a new corporate computing paradigm and even been prophesied to become "a prevailing software-engineering practice, ending the 40-year domination of monolithic software architecture" (Natis 2003). Following this expectation, software producers such as Salesforce or SAP already decided to commercially develop services which are offered on global marketplaces to be reused for on-demand assembling of customized corporate software applications. Such services are expected to build a cornerstone for the success of the SOC concept (Papazoglou 2007), since the service-oriented approach to programming "is based on the idea of composing applications by discovering and invoking network-available services to accomplish some task" (Papazoglou et al. 2007).

Yet, better supporting the SOC paradigm with adequate methods and tools remains a prerequisite for its envisioned breakthrough. To a large extent, its success depends on the ability of consumers to successfully assess offered services and choose the ones best suited for their development project (Mili et al. 1995; Papazoglou et al. 2007; Weyuker 1998). To foster such an assessment, current approaches like the Web services technology enable providers to specify information about the programming interface and the binding information. This allows designers to check for the technical compatibility of services with the rest of the system. However, it remains difficult for business users and designers to understand, which business task a service actually supports, i.e. which business semantics it implements. Without such additional information, service consumers are presumably forced to treat services as "experience goods" (Nelson 1970), whose functionality cannot be adequately assessed against existing requirements until after buying. Where consumers are left unable to discriminate between different goods before buying, the corresponding market is likely to malfunction, though (Akerlof 1970).

In this paper, we propose the new specification language "WS-Functionality" to be used for describing the business semantics of software services in business terms. It draws from the conceptual modeling discipline and uses a system of domain-specific concepts to describe the business semantics implemented by a service. Typical concepts in a description could, e.g., be "warehousing", "random storage", or "first-in-first-out commissioning". From such a description, it becomes clear, that the corresponding service supports warehousing with a chaotic (as opposed to a fixed-bin) storage strategy. Generally, business concepts in our approach are used to define the business tasks that are performed by a service. Concepts can be mapped onto interfaces, methods, inputs, and outputs of a service in order to express their respective business semantics. A system of concepts thus characterizes information items processed by a service, supported business functions, and processes in which a service takes part. From a theoretical perspective, the novel contribution of our approach thereby is to show how conceptual models, which contain such concepts and relationships between them, can be utilized to specify the business semantics of software services.

Taking a design science approach as introduced by Hevner et al. (2004), the WS-Functionality specification language has been iteratively improved and evaluated in different projects until it reached the state presented in this paper. The remaining presentation follows the structure of the design cycle by Takeda et al. (1990) who defined problem awareness, solution suggestion, solution implementation, and solution evaluation as major design steps. In the next section, we firstly discuss related work to confirm the research gap and relate our approach to others. In section 3, we state the problem of assessing the business semantics of software services and define requirements that a possible solution should satisfy. As theoretical solution concept, we then present the meta-model of the WS-Functionality language (section 4). In section 5, we illustrate the language constructs of WS-Functionality. Section 6 focuses on the conducted evaluation of the approach and reports results from an empirical study to validate it. In section 7, we present implications for academia and practice as well as future directions.

Related Work

The description of "classical" reusable software artifacts (like program routines, algorithms, or classes) and of software services in particular has been researched for several years. Resulting approaches aim at documenting "in precise terms the intended effect of a piece of software" (Gehani and McGettrick 1986) and can be distinguished depending on what aspects they focus on. As shown in Table 1, the effect of software artifacts generally is determined by three levels of abstraction (D'Souza and Wills 1999; Olle et al. 1991; Scheer 2000): the *business semantics* expresses the functionality of a software artifact, i.e. the supported tasks and the business context, from a business-oriented viewpoint. It is defined during the conceptual design phase of the development process and described in business terms. The *architecture* of a software artifact describes its programming interface. It provides information about how to technically integrate an artifact into an application system. The architecture is determined during the technical design phase and specified using computer-oriented languages. The *quality* of a software artifact results from its implementation. It is documented by quality characteristics and metrics, e.g. those contained in the ISO 9126 quality model (ISO/IEC 2001). Each of the three abstraction levels can be further structured according to the views of general systems theory (Bertalanffy 1976): the *static view* describes structural properties of a software artifact. Structural properties range from the processed information items and the involved stakeholders (Scheer 2000) to the signature definitions of the programming interface (D'Souza and Wills 1999) to the usability, maintainability, and portability of a software artifact (ISO/IEC 2001). The *functional view* characterizes the capabilities of a software artifact. It gives information about the business tasks and activities that are supported (Scheer 2000), the pre- and post-conditions of interface operations (Beugnard et al. 1999) as well as the provided security, persistency, or transactional capabilities (ISO/IEC 2001). The *dynamic view* states how a software artifact executes at run-time. It comprises the business processes a software artifact supports (Scheer 2000), timing constraints and interactions between interface operations (Beugnard et al. 1999) as well as their reliability and efficiency (ISO/IEC 2001).

Table 1. Abstraction levels of software descriptions

Systems View \ Abstraction level	Business semantics (conceptual design)	Architecture (technical design)	Quality (implementation)
Static view (structure)	Organizations, stakeholders, information items	Signatures (type and interface declarations)	Usability, maintainability, portability
Functional view (capabilities)	Tasks, activities, events	Assertions (pre- and post-conditions, invariants)	Functionality (security, persistency, transactions)
Dynamic view (execution)	Processes, work-flows	Timing constraints (interaction protocols)	Reliability, efficiency

To assess whether a software artifact is reusable in a certain development context, all three abstraction levels have to be analyzed and compared to the requirements specification of the consumer. The requirements specification describes what "software has to do" to be suited for a specific application scenario and, to that end, contains functional as well as quality requirements (IEEE 1998). Functional requirements not only describe desired technical properties such as a certain programming interface. They also prescribe the functionality that a software artifact has to provide from the business user's point of view (IEEE 1998). Approaches to describe the characteristics of "classical" reusable software artifacts, however, solely focus on providing information about their architecture and quality (Beugnard et al. 1999; Han 1998). The majority of these approaches proposes interface description languages which allow specifying the signatures of methods, i.e. the return types of methods, inputs, outputs, and exceptions (D'Souza and Wills 1999). To better describe the effects of the operations provided by the programming interface, there also exist specialized approaches to document pre- and post-conditions, interaction protocols, and quality of service levels as part of a specification (Beugnard et al. 1999; Han 1998). To the best of our knowledge, none of these approaches supports the specification of the implemented business semantics of software artifacts, though.

The current state of the art in the description of reusable software services exhibits a similar situation. To specify the programming interfaces of services, providers can use standardized specification languages such as the Web Service Description Language (WSDL) or the Web Ontology Language for Services (OWL-S). While WSDL only supports the specification of signature lists (Chinnici et al. 2004), OWL-S also allows specifying timing constraints and uses an ontology scheme to achieve a semantically unequivocal description of these architectural properties (Martin et al.

2005). In addition, it provides taxonomies with predefined keywords to classify the application domain of a service. Likewise, approaches such as the Service-Oriented Architecture Modeling Language (SOAML) or the Systems Modeling Language (SysML) allow providers to specify signature lists, assertions, and timing constraints using the UML language (OMG 2008b; OMG 2010). To describe additional service characteristics not covered by such approaches, a whole set of so-called WS-* languages has been proposed (Weerawarana et al. 2005). Among them, WS-Security allows specifying security properties of services like the authentication scheme or the encryption technique. The WS-Policy language provides a notation to document further quality attributes including the reliability or efficiency characteristics of services. The Web Service Semantics (WSDL-S) approach provides a generic mechanism to augment WSDL files with specifications denoted in such languages. Specifically, it addresses the addition of pre- and post-conditions, although no specific format is introduced (Akkiraju et al. 2005). With the approaches mentioned, it is possible to describe both the architecture and quality of services. None of them covers the description of the business semantics as described in Table 1, however.

To document the business semantics of software artifacts, Vitharana et al. (2003) presented an approach that augments programming interfaces with business terms. Specifically, their approach supports attributing interface, method, and data type declarations with keywords to hint at their respective business semantics. In order to facilitate a detailed evaluation of the business semantics of services, however, more comprehensive and interrelated information than just a set of keywords is required. As manually augmenting service descriptions with keywords moreover is a time-consuming task, the authors themselves characterize their approach as “a preliminary step” (Vitharana et al. 2003). To specify business semantics in detail, the Semantics of Business Vocabulary and Business Rules (SBVR) approach proposes a generic meta-model that covers the documentation of vocabularies and business rules (OMG 2008a). Similar to our approach, vocabularies consist of business terms and relationships. In contrast to business rules, which can be formally specified using predicate logic, vocabularies have to be informally described in natural language however. The meta-model mainly introduces facets like “definition”, “synonyms”, or “examples” to structure and harmonize natural language entries into the vocabulary. It does neither completely predefine the semantics of relationship types nor distinguish concept types according to their meaning. Despite their differing characteristics, concepts that denominate things in the business context are hence documented and handled in the same generic way as concepts which refer to business functions. Partly, such restrictions are due to the fact that the meta-model has been designed to document only structural aspects of a business domain like objects and data (Linehan 2008). Since the meta-model moreover is not able to handle structural aspects consistently yet (Linehan 2008), the approach has to be classified as being premature. Its usage in practice is furthermore hampered due to the fact that no specific specification format for describing business vocabularies and rules is introduced.

Comprehensive and interrelated information about business semantics generally can be specified with conceptual business modeling techniques like the ones utilized by the ARIS and IDEF frameworks (Marca and McGowan 2005; Scheer 2000). These frameworks have a very broad scope of application as they address the modeling of businesses and application landscapes in general, though. Accordingly, they make use of a complex mix of notations and usually lack a prescriptive design that predetermines what items exactly have to be modeled in which ways and in which language. We seek to introduce a lightweight, normative approach, however, which endorses the production of comparable descriptions and is tailored to characterize the business semantics of software services. Hence, we introduce a specific language designed to fit into the SOC technology stack. Our solution will, nevertheless, build upon findings and theories of the conceptual modeling domain and its approaches. While we will introduce a language that is able to cover the specification of the business semantics with a single notation, we will deliberately design our solution concept in a way that it can also be used with the notation-mixes of ARIS, IDEF, or the Unified Modeling Language (UML). This can be achieved by defining so-called “language profiles” which limit the set of language constructs and their use according to our solution concept. As we focus on introducing the central solution concept and its evaluation, the design of such additional language profiles is left as a direction of future research.

Problem Statement: What’s in a Service?

Using conventional service description languages as documented above, providers are unable to explicitly specify the business semantics of their services. The lack of such specifications probably makes it difficult for service consumers to select suitable services for their development projects, especially if the business semantics to be assessed is complex and not easy to compare to their functional requirements. To discuss some of these difficulties, Figure 1 shows a quotation with simple functional requirements from a supermarket company. It sought to find a suitable warehouse management service to extend its service-enabled enterprise resource planning application.

While the requirements were provided as free-text, its content is comparable to that contained in more formal specifications (that might, e.g., be created using UML use cases).

We want to use a network-available service to manage our warehouses with up to 5000 storage cells [⓿]. Currently, we operate seven warehouses [❶], each of which is subdivided into receiving, storage, picking, and dispatching areas [❷]. Depending on the area, either a fixed-bin or chaotic storage strategy is applied [❸]. 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 [❹]. A storage bin is located by the numbers of its corridor, rack, and level [❺]. Moreover, storage bins are suited to contain a certain number of either identical articles or standardized pallets [❻]. Standardized pallets, especially those to be shipped to the various stores, can contain up to 20 different articles [❼]. The warehouse management service has to be able to create, read, update, and delete warehouse structures [❽]. 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 [❾]. The stock level of articles has to be managed by accounting services [❿]. (…)

Figure 1. Excerpt of functional requirements in free-text (sample)

Assessing candidate services on the basis of the provided interface descriptions requires service consumers to read and understand specifications which are, e.g., expressed in WSDL. Figure 2 shows a simplified excerpt of a WSDL specification from a candidate service, which supports a warehouse management similar to the one required above. The full specification contains 1501 lines of code in Extensible Markup Language (XML). It defines 31 operations and 25 data types which have to be assessed with respect to their business semantics. To support a better understanding, operations listed in the specification have been documented by the provider with comments using the documentation feature of WSDL. Nevertheless, what likely presents a problem is the fact that potential service consumers have to derive the implemented business semantics from the specification, since it is covered implicitly only. To determine whether the service allows managing warehouses in which the storage strategy varies from area to area, assessors have to inspect line 20 of the specification. From there, they can presume that the strategy can be configured per storage bin when using the »createStorage« operation to model a new warehouse. This would satisfy user requirement ❸ (see Figure 1). Contrary to requirement ❼, storage bins appear to support a maximum of 10 article counts only (line 19). Furthermore, the candidate service seems to disallow modeling separate areas for receiving and dispatching as required by ❷. Instead, it appears to support the creation of shipping areas as a more general concept (line 13). Line 29 implies that only a maximum of five warehouses is supported since the operation »getStorageIds« returns an array with a capacity of five. This limitation would violate requirement ❶. Features like the commissioning strategy, which cannot be customized using the interface, are not mentioned at all. Consequently, it remains unclear whether the candidate service supports the FEFO strategy as prescribed by requirement ✪.

Since educated guesses as the ones described above provide in fact just indications, consumers cannot be sure about the truly provided business semantics of services without obtaining additional information. To support getting a deep-level understanding, we believe that a service description must go beyond annotations as provided with the sample WSDL file. Instead, the documentation must be powerful enough to represent the relevant aspects of the implemented business semantics (Wand and Weber 1993). A language to describe the business semantics of services hence has to fulfill requirement *R1 (completeness): the language must be able to describe the implemented business semantics in detail*. Since we document the business semantics of a service in order to make it efficiently assessable by consumers, the language should provide a preferably small set of non-redundant constructs. In general, this will enhance the consumer's ability to become familiar with the language and use it properly (Wand and Weber 1993). The language hence should satisfy requirement *R2 (clarity): it has to provide a compact, unambiguous set of language constructs*. Guidelines on how to use these constructs for the description will furthermore reduce variations among service documentations (Hadar and Soffer 2006) and allow consumers to better compare them. Hence, we postulate requirement *R3 (strictness): the language has to establish normative description guidelines*.

When analyzing WSDL specifications, consumers are distracted from understanding the business semantics by the way in which relevant information is represented. Statements that contain information about the implemented semantics are expressed in an abstract, technical way that is not straightforward to understand. E.g., line 29 of the WSDL file depicted in Figure 2 states »`<s:element minOccurs="0" maxOccurs="5" name="getStorageIds" type="s:string"/>`«. Specifying that »the number of manageable warehouses is limited to a maximum of five« is not less precise but probably easier to understand. The language to specify the business semantics of services thus has to fulfill requirement *R4 (understandability): it has to be readily understandable for service consumers*. In addition, a WSDL documentation intermixes aspects of the business semantics with implementation aspects such as type definitions, message definitions etc. To keep these two aspects separate, we formulate requirement *R5 (technology independence): the language must describe the business semantics independently of its software-*

technical realization. Yet, the business semantics of a software service needs to be related to the programming interface. To document by which operations the business semantics is implemented, we state requirement *R6 (technology reference): the language has to relate the business semantics to elements of the programming interface*. Finally, a language to specify the business semantics of services will have to integrate into the current technology to be relevant for research and practice. We therefore want it to comply with requirement *R7 (technology integration): the language has to integrate into the SOC technology stack*.

```
<wsdl:definitions>
  <wsdl:types>
    <s:schema targetNamespace="http://www.sample.storage-management.org/">
      <s:element name="StructureStorage"> <s:complexType> <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="name" type="s:string"/>
        <s:element minOccurs="1" maxOccurs="unbounded" name="storageareas" type="StructureStorageArea"/>
      </s:sequence> </s:complexType> </s:element>
      <s:element name="StructureStorageArea"> <s:complexType> <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="type" type="tns:AreaType"/>
        <s:element minOccurs="1" maxOccurs="750000" name="bins" type="Bin"/>
      </s:sequence> </s:complexType> </s:element>
      <s:simpleType name="AreaType"> <s:restriction base="s:string">
        <s:enumeration value="storing"/> <s:enumeration value="picking"/> <s:enumeration value="shipping"/>
      </s:restriction> </s:simpleType>
      <s:element name="Bin"> <s:complexType> <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="typeId" type="s:string"/>
        <s:element minOccurs="1" maxOccurs="1" name="slot" type="s:int"/>
        <s:element minOccurs="1" maxOccurs="1" name="level" type="s:int"/>
        <s:element minOccurs="0" maxOccurs="10" name="articleCounts" type="ArticleCount"/>
        <s:element minOccurs="1" maxOccurs="1" name="strategy" type="tns:Strategy"/>
      </s:sequence> </s:complexType> </s:element>
      <s:element name="ArticleCount"> <s:complexType> <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="id" type="s:string"/>
        <s:element minOccurs="1" maxOccurs="1" name="count" type="s:int"/>
      </s:sequence> </s:complexType> </s:element>
      <s:simpleType name="Strategy"> <s:restriction base="s:string"> <s:enumeration value="static"/> <s:enumeration value="chaotic"/>
      </s:restriction> </s:simpleType>
      <s:element name="getStorageIds"> <s:complexType> <s:sequence>
        <s:element minOccurs="0" maxOccurs="5" name="getStorageIds" type="s:string"/>
      </s:sequence> </s:complexType> </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="createStorageSoapIn"> <wsdl:part name="parameters" element="tns:StructureStorage"/> </wsdl:message>
  <wsdl:message name="createStorageSoapOut"> <wsdl:part name="parameters" element="tns:createStorageResponse"/> </wsdl:message>
  <wsdl:message name="getStorageIdsSoapIn"> <wsdl:part name="parameters" element="tns:getStorageIdsInput"/> </wsdl:message>
  <wsdl:message name="getStorageIdsSoapOut"> <wsdl:part name="parameters" element="tns:getStorageIds"/> </wsdl:message>
  <wsdl:portType name="Storage Management Services">
    <wsdl:operation name="createStorage"> <wsdl:documentation>Role: Storage Management. Description: Uses a storage structure
      to create a new storage. Inputs: datatype StructureStorage. Output: new storage ID.</wsdl:documentation>
      <wsdl:input message="tns:createStorageSoapIn"/> <wsdl:output message="tns:createStorageSoapOut"/>
    </wsdl:operation>
    <wsdl:operation name="getStorageIds"> <wsdl:documentation>Role: Storage Management. Description: Gets IDs of existing storages.
      Output: storage IDs (arraySize <= 5).</wsdl:documentation>
      <wsdl:input message="tns:getStorageIdsSoapIn"/> <wsdl:output message="tns:getStorageIdsSoapOut"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Figure 2. Simplified WSDL excerpt of a candidate service (sample)

We expect a language that allows specifying the business semantics of services and fulfills the requirements as discussed above to *enhance the usability of service descriptions* for consumers during the service assessment phase. The usability of service descriptions consists of three aspects (Frøkjær et al. 2000; ISO/IEC 1998): the effectiveness (i.e. accuracy), efficiency (i.e. effectiveness in relation to effort), and satisfaction (i.e. users' comfort) of consumers during the assessment of service descriptions with the goal to identify appropriate services fulfilling their functional requirements. In particular, we make the following propositions when using WS-Functionality:

P1 (Increased Effectiveness): We expect the ability of consumers to identify appropriate services to increase.

P2 (Increased Efficiency): We expect the ability of consumers to identify appropriate services in relation to the time needed to increase.

P3 (Increased Satisfaction): We expect the comfort of consumers with the service description to increase.

Solution Concept: A Meta-Model to Specify the Business Semantics of Services

For the WS-Functionality language to specify the business semantics of services, we implemented a modular design as illustrated in Figure 3. First of all, we developed a meta-model that contains normative prescriptions regarding the content of a *specification*. Thereafter, we designed the language constructs that determine the *presentation* of such specifications. By distinguishing the two design-steps, we were able to separate the theoretical solution concept, i.e. the meta-model, from the solution implementation, i.e. the actual language constructs. Furthermore, it becomes possible to allow additional presentation formats, e.g. to support different target groups and modeling frameworks.

The meta-model builds upon the design of conceptual models to capture the business semantics of software services. In general, the semantics underlying a business domain is expressed by its particular system of concepts (Bunge 1977). The system of concepts characterizes the meaning of domain aspects such as exchanged information, actions of stakeholders, work flows, incidents etc. (Scheer 2000). It consists of two parts: a *vocabulary* with technical terms that denominate *concepts* from the domain, and a compilation of *statements* which define *relationships* between concepts to express complex domain semantics (Fensel 1998; Gómez-Pérez et al. 2004). For the warehousing domain, characteristic technical terms would be »warehouse«, »warehouse area«, »article«, »storage strategy«, »commissioning strategy« etc. Typical statements that interrelate terms to express complex domain semantics are e.g. »a warehouse consists of numerous warehouse areas« and »a warehouse area is governed by a storage strategy«.

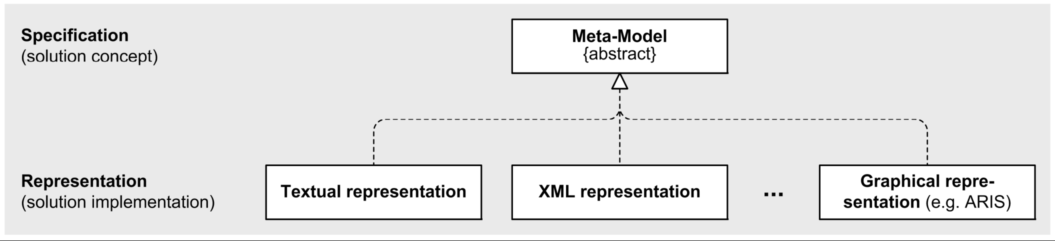


Figure 3. Modular architecture of WS-Functionality (depicted using UML)

To improve the communication among business users and designers and allow them to better understand the business semantics of a domain, the vocabulary and concept relationships are often documented explicitly as a conceptual model. Conceptual modeling is defined as “the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication” (Mylopoulos 1992). The resulting conceptual model provides “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). These semantics are described independently of programming technologies and implementation considerations (Hadar and Soffer 2006; Topi and Ramesh 2002). For the design of WS-Functionality, we reuse the conceptual modeling approach and specialize it to document the business semantics of software services.

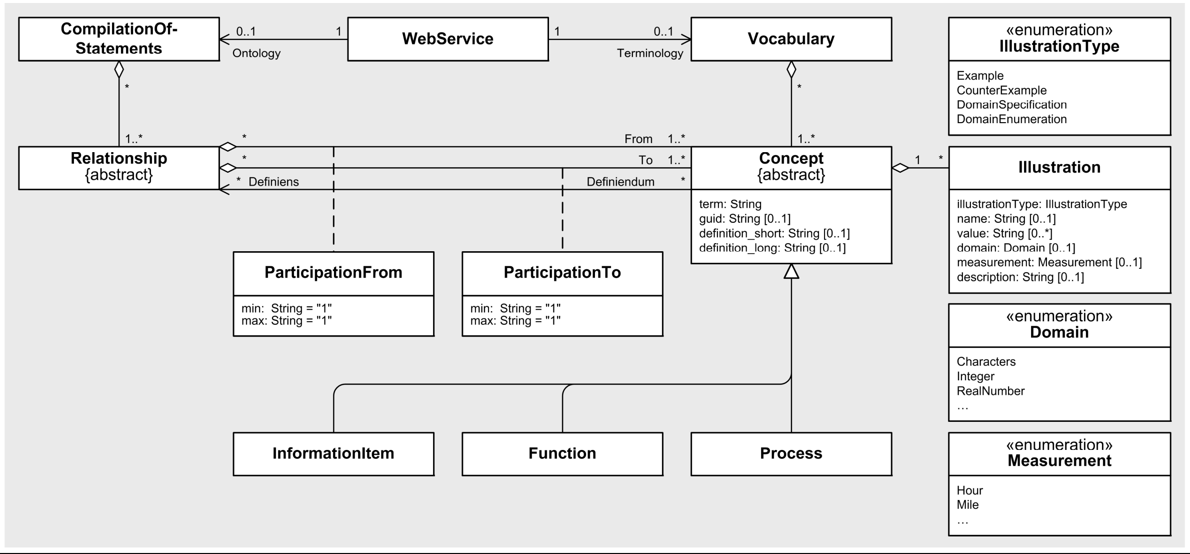


Figure 4. Formal meta-model to specify a service-specific system of concepts (depicted using UML)

As we do not aim at documenting entire domains, but less complex software services, we firstly reduced the set of concept types to build a lightweight ontology of services. From the corresponding literature, we can conclude that the business semantics of services is characterized by three concept types: the *information items* that are processed, the *functions* which are performed, and the *processes* that a service’s functions can be integrated into (Bunge 1977; Hadar and Soffer 2006; Lyons 1977; Scheer 2000). Information items represent (data) objects which are subject to information processing. Functions correspond to business tasks which are automated or at least supported by the implemented software service. Processes stand for complex business activities which are realized by a collaboration of temporally orchestrated functions. A warehouse management service, e.g., may process information items such as delivery notes, commissioning plans, or purchase orders. It may provide functions to create, read, update, and delete warehouse structures, to generate and execute commissioning plans, as well as to update stock levels. These functions can be integrated into the delivery, shipment, and accounting processes of companies.

The three concept types form the major building blocks of the domain vocabulary (see Figure 4). Furthermore, they characterize the programming interface of services. As a consequence, some of the information items can directly be mapped onto type definitions or parameters, while some of the business functions can be mapped onto operations. Probably, the vocabulary will also contain concepts not directly related to items of the programming interface. Such concepts describe the semantics of interface items in more detail and set it into the relevant context. For each concept in the vocabulary, its identifying term has to be specified (see Figure 4). It can be complemented with a globally unique id (a machine-readable identifier), a definition to specify its meaning informally, and illustrations to show concrete representations. Some concepts may furthermore refer to auxiliary elements of the WS-Functionality language such as actors, who perform functions, or events, which trigger process steps (Bunge 1977; Scheer 2000).

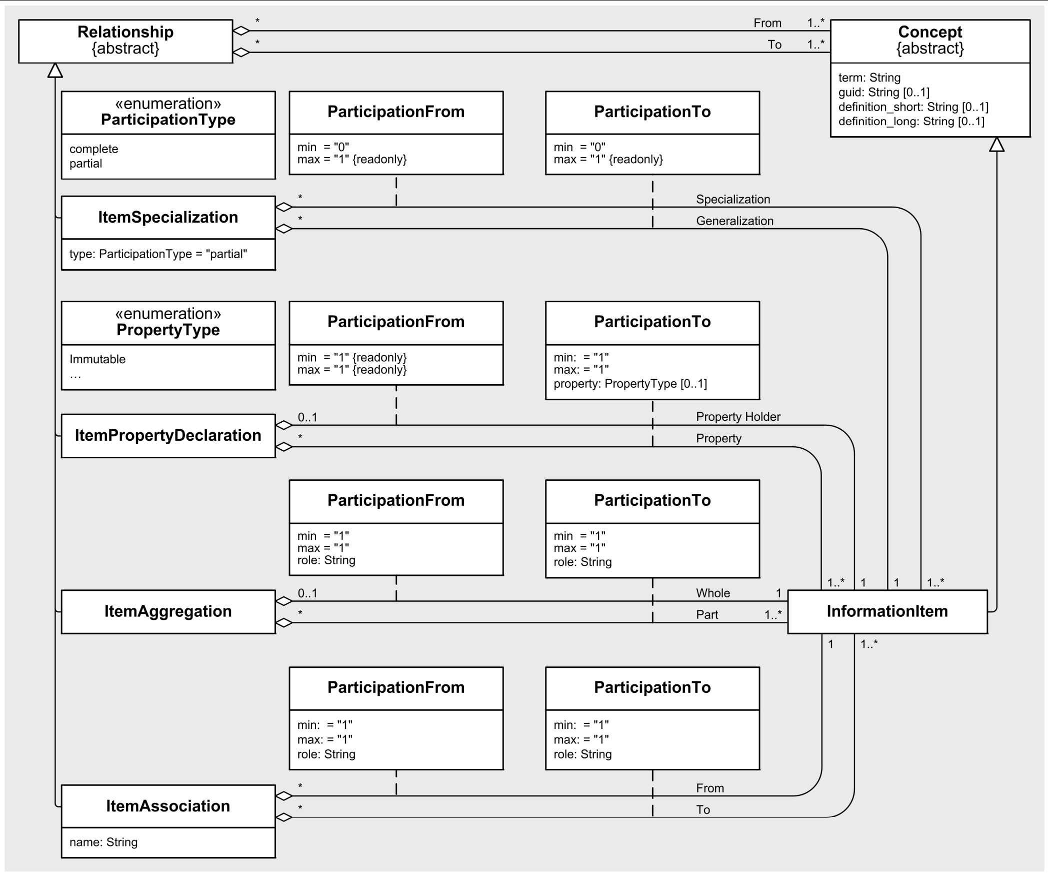


Figure 5. Relationship types to connect information items (depicted using UML)

In order to express complex aspects of a service's business semantics, statements can be articulated on the basis of the domain vocabulary. From a linguistic perspective, such statements define relationships between the concepts contained in the vocabulary. In contrast to natural language, we limited the relationship types that each concept type can participate in. Therefore, we introduced a set of relationship types with predefined semantics and attributes (such as roles for connected concepts, cardinalities etc.). Statements hence cannot be defined arbitrarily, but only according to the prescriptions given in the meta-model. From the corresponding literature, we can conclude that information items, functions, and processes are basically interrelated by five types of relationships: *specialization* *relationships*, *property declarations*, *aggregations*, *associations*, and temporal *flows* (Bunge 1977; Hadar and Soffer 2006; Lyons 1972).

Specialization relationships connect concepts of the same type and express a conceptual subsidiarity between them, i.e. one concept is more specific than another (Lyons 1972). Example statements to describe a warehousing service which can be specified with WS-Functionality are: »a picking area is a special warehouse area« and »an article is a special storage item«. Specialization relationships constitute so-called concept hierarchies. Such hierarchies may exist either between information items (see Figure 5) or between business functions.

Property declarations define concepts to be dependent on another concept and assign them as properties to this concept. Properties can be viewed as second-class concepts which are integral parts of another concept (Bunge 1977). Mostly, property declarations occur between information items (see Figure 5). Example statements to characterize a warehousing service could be: »a storage bin has a loading capacity« and »a warehouse area has a storage strategy«. Properties contain simple values which are either specified by an enumeration or by determining a domain: »a storage strategy is either fixed-bin or chaotic«, »a loading capacity is a real number and measured in pounds«. Accordingly, we predetermined a set of domains and measuring units (see Figure 4). Property declarations also occur between functions and information items in order to assign inputs and outputs to a function (see Figure 6). For a warehousing service, we can e.g. specify: »A stockkeeper« (actor) »commissions« (function) »one or more articles« (characteristic item/input) »on the basis of a commissioning plan« (input) »to a shipment« (output). As can be seen from the example, functions always operate on a characteristic information item. To characterize a function, its term (a verb) must be specified together with the characteristic item (i.e. »commission articles« instead of just »commission«). Besides additional in- and outputs, we can also assign actors who perform a business function. Currently, restrictions apply regarding the specification of conditions, under which a business function may or may not be executed. Such conditions currently can only be specified informally as part of the concept definition, as we left a more formal specification as a future research direction.

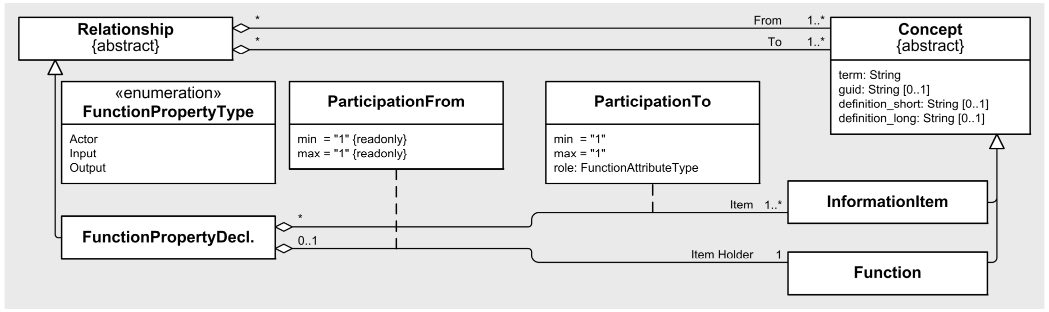


Figure 6. Property declaration relationship type for functions (depicted using UML)

Aggregations associate either information items or functions to form a new, composite information item or function (Bunge 1977). The composite item may have emergent properties which are not possessed by the constituent items (see Figure 5). In contrast to a property declaration, an aggregation combines concepts which stand for themselves. Used to relate functions, this relationship type supports a stepwise refinement (Wirth 1971) of complex business functions into more elementary operations that can be offered at a service interface. Examples for aggregations are »a warehouse consists of one or more warehouse areas«, »a warehouse area consists of one to 20 storage bins«, or »pick an article consists of create a commissioning plan, execute a commissioning plan, and update a stock level«.

Associations are generic ways to relate information items without detailing on the effect of this relation (Hadar and Soffer 2006). Since both the semantics and the name of such associations cannot be predefined (see Figure 5), they should only be used if the pre-mentioned relationship types are inappropriate.

Flows describe temporal courses of business functions. Accordingly, they can be used to specify how complex business activities (processes) are executed step by step. This dynamic aspect is complementary to a static functional decomposition as described by aggregation relationships. The flow relationship supports the specification of simple workflow patterns, i.e. sequences, parallel flows, and inclusive / exclusive branches of functions (van der Aalst and Kumar 2003; van der Aalst et al. 2003). Using flows, we can e.g. describe the process to pick an article: »create a commissioning plan precedes execute a commissioning plan« and »execute a commissioning plan precedes update a stock level«. Thereby, each function can be attributed with flow-specific properties like triggering events (see Figure

7). Note that the flow relationship is not as expressive as the constructs of sophisticated process modeling languages. It has deliberately been designed to allow specifying simple workflows to support getting an understanding of causal dependencies between functions. We use such process descriptions primarily to describe the order in which service operations should be invoked according to the underlying business semantics. Often, this order is not obvious as software services usually provide numerous operations at their programming interfaces.

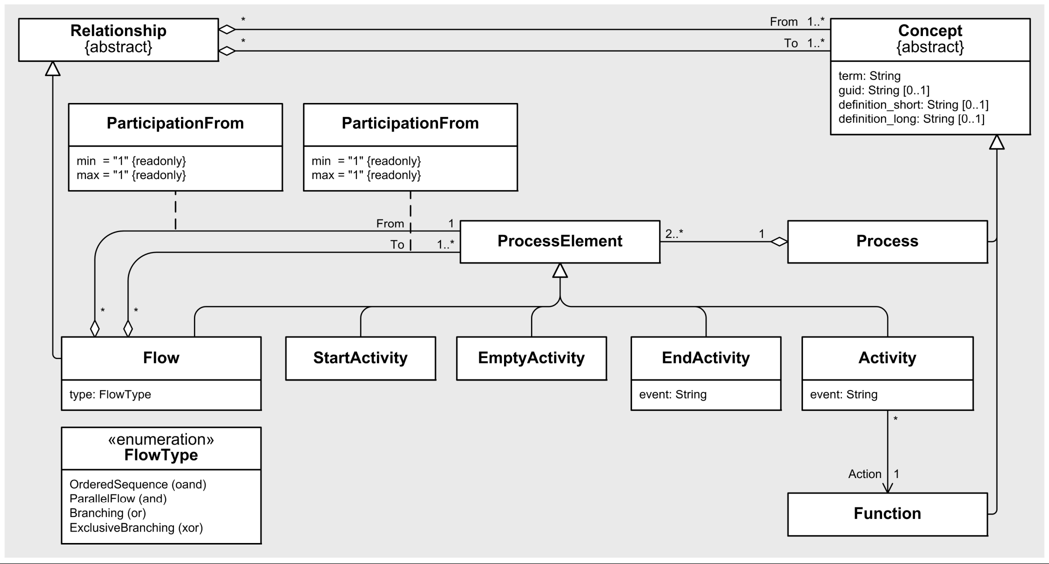


Figure 7. Flow relationship type (depicted using UML)

To specify the business semantics of services, we use only *three concept types* (information items, functions, processes) and *five relationship types* (specialization relationships, property declarations, aggregations, associations, and flows). Compared to conceptual modeling approaches like ARIS or IDEF, this approach thus provides a very compact formalism. As each construct is used to specify a different domain aspect and each has a formally defined semantics, our solution concept supports requirement *R2 (clarity)*. The solution concept further contains normative description guidelines as required by *R3 (strictness)*, since it allows connecting the language constructs in pre-defined ways only. Note that we could not discuss all provided guidelines for reasons of brevity: additional guidelines (e.g. that a process must have at least one start and at least one end activity) are specified as formal constraints that complement the depicted meta-model excerpt. As a result of building on conceptual modeling theories that are technology-agnostic by definition, we satisfy *R5 (technology independence)*. To fulfill *R6 (technology reference)*, we support mapping concepts to pieces of WSDL interfaces: information items may link to type definitions and parameters, functions may link to operations, and processes may link to port types. The main goal was, however, to specify the business semantics in detail as required by *R1 (completeness)*. While it is hard to prove completeness, we have tested and iteratively improved the meta-model in case studies until it reached the presented state. It allows specifying considerably more details than related approaches (see section 2) and, as exemplified above, is able to express complex domain semantics to relate a service interface to its business context.

Solution Implementation: The WS-Functionality Language

To denote the business semantics of software services, we designed a text-based language as presentation format. It captures all concept and relationship types as predetermined in the WS-Functionality meta-model. The reasons why we decided to propose a text-based notation are threefold: first of all, we wanted designers and business users to be able to straightforwardly read the described business semantics. The text-based presentation format was therefore based on the English natural language. It provides generic sentence patterns to describe the business semantics of services as human-readable statements (see Figure 8). Compared to natural language, however, the introduced presentation format is much more restrictive as it allows only statements conforming to the provided sentence patterns. This facilitates the homogeneity and comparability of descriptions since they only contain relationships between concepts as pre-defined by the meta-model. In addition, the restrictive language is more precise than ordinary natural language because the semantics of all statements is clearly defined (Ortner and Schienmann 1996).

It will consequently be easier for humans to understand the business semantics provided by a service correctly. We can furthermore use such standardized specifications for automated processing and compatibility tests of service descriptions in future work. Ideally, the standardization therefore should go beyond the relationships and also cover the vocabulary, however. To that end, we support the introduction and usage of so-called controlled vocabularies (Prieto-Díaz 2003) together with the WS-Functionality language. We did not make controlled vocabularies a mandatory part of the language, however, since this would only limit its application. Instead, we will later on present empirical indications that WS-Functionality is able to improve the comparison and selection of services even in scenarios where the vocabulary slightly differs between providers and consumers. Thereby, the required explicit definition of business terms specifically allowed human users to compare and match differing denominations.

Specialization	
Items:	(A|An) *SpecializedItem$_1$* (**or** (a|an) *SpecializedItem$_2$*)$^+$ **is** (a|an) *GeneralizedItem*.
Functions:	(A|An) *SpecializedFunction$_1$* (**or** (a|an) *SpecializedFunction$_2$*)$^+$ **is** (a|an) *GeneralizedFunction*.
Aggregation	
Items:	(A|An) *AggregateItem* [**as** *Role$_1$*] **is composed of** (a|an|[*Min$_1$* **to**] (*Max$_1$*|**many**)) *PartItem$_1$* [**as** *Role$_2$*] (**and** (a|an|[*Min$_2$* **to**] (*Max$_2$*|**many**)) *PartItem$_2$* [**as** *Role$_3$*])$^+$.
Functions:	(A|An) *AggregateFunction$_1$* [**as** *Role$_1$*] **is composed of** (a|an|[*Min$_1$* **to**] (*Max$_1$*|**many**)) *PartFunction$_1$* [**as** *Role$_2$*] (**and** (a|an|[*Min$_2$* **to**] (*Max$_2$*|**many**)) *PartFunction$_2$* [**as** *Role$_3$*])$^+$.
Property Declaration (Items)	
Main:	(A|An) *Item$_1$* **has** (a|an|[*Min$_1$* **to**] (*Max$_1$*|**many**)) *PropertyItem$_1$* (**and** (a|an|[*Min$_2$* **to**] (*Max$_2$*|**many**)) *PropertyItem$_2$*)$^+$.
Options:	[(A|An) *PropertyItem* **has unchanging values**.]
Values:	[((A|An) *PropertyItem* **is** *PropertyValue$_1$* (**or** *PropertyValue$_2$*)*.) | ((A|An) *PropertyItem* **is** (a|an) *Domain* [**and is measured in** *MeasuringUnit*].)]
Property Declaration (Functions)	
Characteristic Item:	(((A|An) *Actor*)|**Someone**) **does** *FunctionTerm* (a|an|[*Min$_1$* **to**] (*Max$_1$*|**many**)) *CharacteristicItem*
Additional Inputs:	[**with** (a|an|[*Min$_2$* **to**] (*Max$_2$*|**many**)) *Item* (**and** (a|an|[*Min$_3$* **to**] (*Max$_3$*|**many**)) *Item*)*]
Additional Outputs:	[**to** (a|an|[*Min$_4$* **to**] (*Max$_4$*|**many**)) *Item* (**and** (a|an|[*Min$_5$* **to**] (*Max$_5$*|**many**)) *Item*)*].
Association	
Main:	(A|An|[*Min$_1$* **to**] (*Max$_1$*|**many**)) *Item$_1$* [**as** *Role$_1$*] (**is**|**are**) **in** (a|an) *RelationshipName* **relationship with** (a|an|[*Min$_2$* **to**] (*Max$_2$*|**many**)) *Item$_2$* [**as** *Role$_2$*] (**and** (a|an|[*Min$_3$* **to**] (*Max$_3$*|**many**)) *Item* [**as** *Role$_3$*])$^+$.
Flow and Process-Attribute Declaration	
Start Activities:	*ProcessTerm* **starts with** *Element$_1$* (**or** *Element$_2$*)*.
Sequence:	*Element$_1$* **precedes** *Element$_2$*.
Parallel:	*Element$_1$* **precedes begin of parallel execution**. **Begin of parallel execution precedes** *Element$_2$* (**and** *Element$_3$*)$^+$. […] *Element$_4$* (**and** *Element$_5$*)$^+$ **precede end of parallel execution**.
Inclusive:	*Element$_1$* **precedes begin of branched execution**. **Begin of branched execution precedes** *Element$_2$* (**or** *Element$_3$*)$^+$. […] *Element$_4$* (**or** *Element$_5$*)$^+$ **precedes end of branched execution**.
Exclusive:	*Element$_1$* **precedes begin of conditional execution**. **Begin of conditional execution precedes either** *Element$_2$* (**or** *Element$_3$*)$^+$. […] **Either** *Element$_4$* (**or** *Element$_5$*)$^+$ **precedes end of conditional execution**.
Elements:	*Function* [**on** *Event*] (Activity); **do nothing** (Empty Activity); **terminate execution** [**with** *Event*] (End Activity)

Figure 8. Sentence patterns to describe the business semantics in WS-Functionality

Literature provides a second reason to introduce a text-based format, as there are indications that such a format might be superior to graphical representations regarding its usability during the assessment of software services. Amongst others, Mayer et al. (1990) empirically examined the impact of text versus diagrams on a user's ability to learn a domain. They argue that users who receive text will outperform those who receive diagrams in tasks based on verbatim retention. Verbatim retention is especially relevant during the assessment of services, where their business semantics is analyzed, verified against corresponding requirements and compared to the descriptions of alternative candidates. The third reason for creating a text-based format was that service description languages used in practice (such as WSDL, WS-Security, WS-Transaction etc.) are usually based on text as they are displayed and exchanged on Web platforms such as e.g. service marketplaces. Since we wanted to integrate into the existing technology stack, we deliberately chose to also make use of a text-based format.

Using the WS-Functionality presentation format, service providers can specify a vocabulary of business concepts in a first step (see Figure 9). Building upon the defined concepts and the provided sentence patterns, they can then formulate statements, which relate the concepts to each other and so describe the business semantics implemented by the offered service. In a third step, service providers can relate concepts of the business vocabulary to elements of the programming interface in order to depict which elements of the programming interface implement a certain business semantics. To assess provided services, consumers should first of all analyze their business semantics. Processing the information described using WS-Functionality will thereby help consumers to get an understanding of the business semantics that a service provides. With such an understanding, they become able to verify the provided business semantics against the one documented in their requirements specification. Once the business semantics of a service matches the requirements, consumers will then need to analyze its programming interface in order to determine the technical compatibility of the service with the rest of the system. Ideally, consumers will furthermore be able to analyze the quality characteristics of a service to judge if it fulfills existing non-functional requirements. To analyze the programming interface of a service, consumers have to process additional specification documents such as a WSDL file. Since the WSDL will hence be used in addition to WS-Functionality, we support a direct mapping of business concepts to elements of a WSDL document as illustrated in Figure 9 (III).

I. Vocabulary

warehouse	A warehouse is a commercial facility to stock articles. To function efficiently, the facility must be properly slotted.
stockkeeper	A stockkeeper is tasked with maintaining warehouses. Responsibilities generally include procurement and shipping.
warehouseArea	A warehouse area is a sector that is dedicated to carry out a specific warehousing task.
storageBin	A bin is a physical place which buffers articles.
…	

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 [⓿]. A warehouseArea **has** an areaType. An areaType **is** storing **or** picking **or** shipping [❷]. A storageBin **has** a binType **and** a slot **and** a level [❺] **and** 1 **to** 10 articleCounts [❼] **and** a storageStrategy. A storageStrategy **is** static **or** chaotic [❸]. A binType **has** a height **and** a width **and** a depth **and** a tonnageCapacity [❹] **and** a storageUnit. A storageUnit **is** standardPallet **or** singleArticle [❻]. 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 [❶]. A stockkeeper **does** create a commissioningPlan **with** a customerOrder. A stockkeeper **does** create a storagePlan **with** a deliveryReceipt [❾]. A stockkeeper **does** book an inventoryChange **with** a reservation [❿].

An administrator **does** create a warehouse [❽].

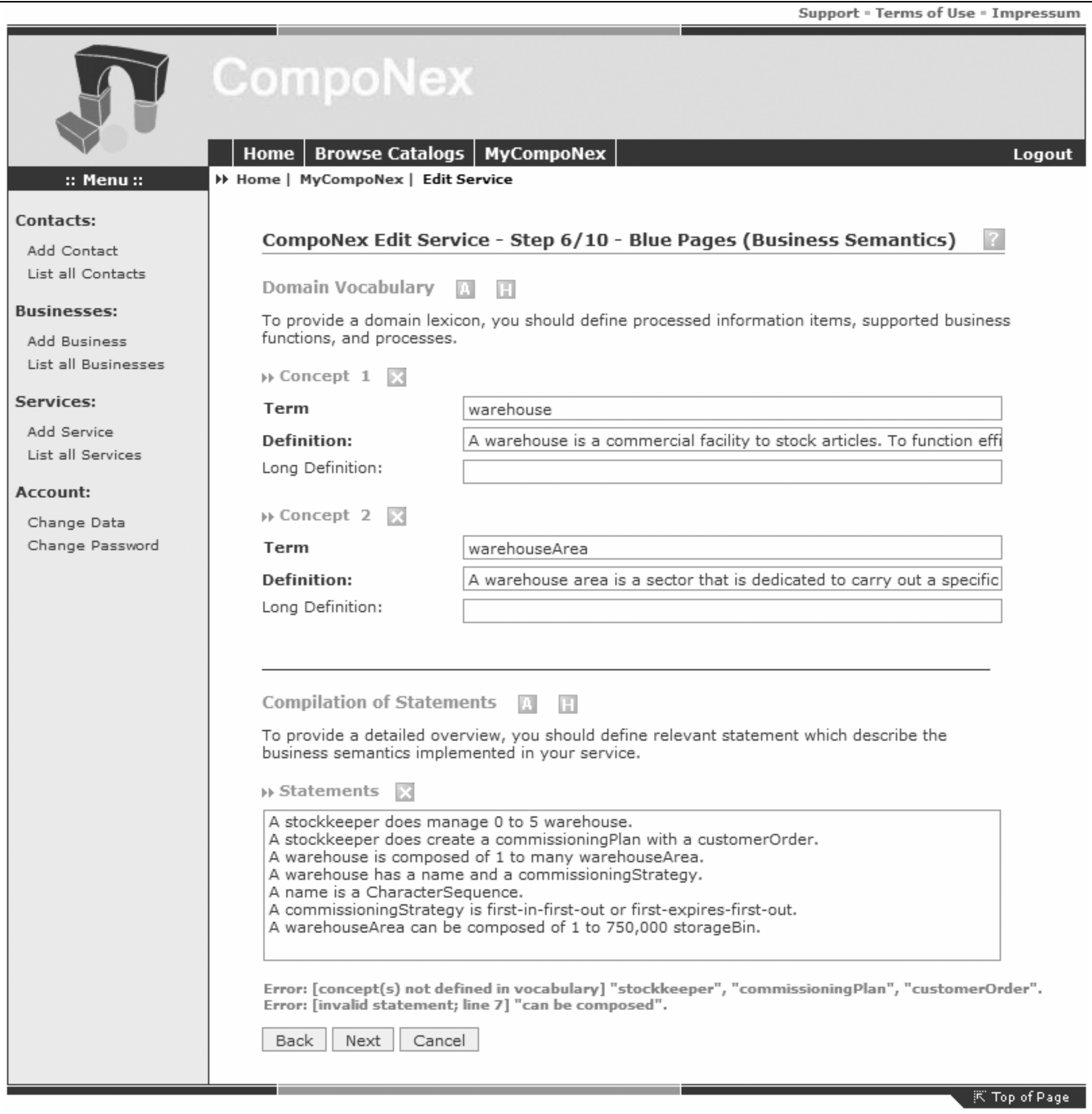
III. Interface Mapping

warehouse	StructureStorage
warehouseArea	StructureStorageArea
storageBin	Bin
create warehouse	Storage Management Services::createStorage
get warehouseArea	Storage Management Services::getStorageAreas
…	

Figure 9. Exemplary statements to specify the business semantics of software services

Figure 9 shows the statements that have to be described in WS-Functionality in order to document the business semantics of the service which was introduced during the problem statement earlier on (see Figure 2). Compared to its WSDL file, which contains 1501 lines of code in XML, the description of the business semantics in WS-Functionality is much more compact. The specification depicted in Figure 9 furthermore explicitly allows getting an understanding of the implemented business semantics and matching it with existing requirements. As the numbering denoted in squared brackets illustrates, all existing requirements can be analyzed against the description denoted in WS-Functionality. Using WS-Functionality, it is even possible to specify aspects of the implemented business semantics (such as the commissioning strategy) which have no representation at the programming interface. Such aspects could not be evaluated on the basis of the WSDL file at all. For reasons of brevity, both the vocabulary and the interface mapping had to be shortened in Figure 9 however. A detailed case study that illustrates how to specify the business semantics can be obtained from the authors.

As discussed in the previous section, WS-Functionality already satisfies requirements *R1 (completeness)*, *R2 (clarity)*, *R3 (strictness)*, *R5 (technology independence)*, and *R6 (technology reference)* due to its meta-model. The presentation format introduced in this section moreover uses a notation that is understandable for business users and designers. The design of WS-Functionality hence fulfills *R4 (understandability).* Since the format contains a mapping onto WSDL constructs, WS-Functionality furthermore integrates into the SOC technology and so supports *R7 (technology integration).* To assess the technical feasibility of the designed specification language, we implemented a prototypic compiler that is able to verify specifications denoted in the presentation format. We integrated this compiler into an electronic marketplace to trade software services and implemented functionality to describe and display information about the business semantics of provided services (see Figure 10). The marketplace is based on the Microsoft .NET platform and described in detail elsewhere (Overhage and Schlauderer 2010). Its prototype can be accessed at www.componex.biz. Regarding the chosen design science approach, the implemented compiler and its application as part of a service marketplace serves as a first technical proof of concept (Hevner et al. 2004). Besides, we analyzed the applicability of our language by repeatedly specifying software services for the business domain together with partners from software industry. E.g., we used the WS-Functionality language in two larger case studies in which we were able to specify warehouse management and flight booking services. Experiences gathered during such case studies helped us to identify shortcomings of our concept and provided the basis for improving our solution in various iterations.



CompoNex Edit Service - Step 6/10 - Blue Pages (Business Semantics)

Domain Vocabulary

To provide a domain lexicon, you should define processed information items, supported business functions, and processes.

Concept 1

Term	warehouse
Definition:	A warehouse is a commercial facility to stock articles. To function effi
Long Definition:	

Concept 2

Term	warehouseArea
Definition:	A warehouse area is a sector that is dedicated to carry out a specific
Long Definition:	

Compilation of Statements

To provide a detailed overview, you should define relevant statement which describe the business semantics implemented in your service.

Statements

```
A stockkeeper does manage 0 to 5 warehouse.
A stockkeeper does create a commissioningPlan with a customerOrder.
A warehouse is composed of 1 to many warehouseArea.
A warehouse has a name and a commissioningStrategy.
A name is a CharacterSequence.
A commissioningStrategy is first-in-first-out or first-expires-first-out.
A warehouseArea can be composed of 1 to 750,000 storageBin.
```

```
Error: [concept(s) not defined in vocabulary] "stockkeeper", "commissioningPlan", "customerOrder".
Error: [invalid statement; line 7] "can be composed".
```

Figure 10. Specification of Business Semantics with WS-Functionality (Prototype)

Evaluation: How Usable is WS-Functionality?

Assuring the technical feasibility and general applicability of our solution provided the basis to evaluate, whether WS-Functionality is indeed able to increase the usability of service descriptions during the evaluation of services as claimed in section 3. To test our proposition, we conducted a laboratory experiment which involved (i) identifying appropriate service candidates that fulfill a given set of functional requirements as well as (ii) comparing service descriptions and determining differences in the business semantics. The main factor examined in the experiment was the language used to describe services. Figure 11 depicts the design of the experiment. We used materials from a preceding case study in the warehouse management domain to derive a requirements specification and a total of seven service descriptions. The experiment started with an ex-ante interview on SOC knowledge, whose results were used to identify and exclude possible outliers afterwards. Thereafter a requirements specification document, which was the basis for the first task, was handed out to the participants. All of them had the same time to go through the document and identify requirements carefully. The participants were then randomly assigned to two equally large groups and either used WSDL files only (control group) or a combination of WSDL files and descriptions of the business semantics, which were denoted in WS-Functionality language (treatment group). The first task comprised comparing the business semantics of services to functional requirements to identify appropriate service candidates. At the end of the first task, participants recorded whether the service description allowed them to come to a conclusion and next marked selected service(s). During the second task, participants had to compare service descriptions in order to determine differences in the implemented business semantics. Participants were asked to denote as many differences as they could find. There was no time restriction for both tasks, however time was recorded for the following analysis. A survey on the satisfaction of the participants completed the experiment.

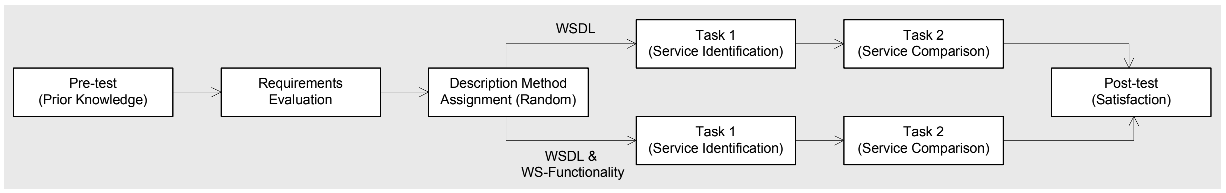

WSDL
Pre-test (Prior Knowledge)
Requirements Evaluation
Description Method Assignment (Random)
Task 1 (Service Identification)
Task 2 (Service Comparison)
Post-test (Satisfaction)
Task 1 (Service Identification)
Task 2 (Service Comparison)
WSDL & WS-Functionality

Figure 11. Design of experiment

Participants in the experiment were 32 graduate students from business administration, information systems, and computer science degree programs. Each of the students had information systems as a major field of study. The control group contained four business administration, five information systems, and seven computer science students. The treatment group consisted of three business administration, five information systems, and eight computer science students. Among the participants were 30 men and two women, who were in between their third and sixth academic year. All attended a course in component & service engineering and voluntarily participated in the experiment. Following Batra et al. (1990), the only incentive offered to the students was an in-depth training in service evaluation techniques, yet we still observed a high motivation. Ex-ante interviews showed that they had a slightly differing knowledge about SOC, but none was a freshman. Although using information systems students as substitutes for business analysts and designers may be questionable, we followed such a design as literature provides good arguments for the selection of students instead of practitioners in controlled settings. Most importantly, practitioners are likely to (at least implicitly) bring in their own experiences and service evaluation procedures, which might bias the results (Gemino and Wand 2004). It is hence easier to ensure the application of a specific methodology with novices than it is with experts who already automated “their problem-solving processes to the point at which they are no longer able to articulate what they are doing” (Anderson 1982; Vessey and Conger 1994).

During the execution of the experiment, several documents were provided to the participants, which are described below. Due to the interest of brevity we cannot depict all of them in the paper, however[1]. The *pre-test* consisted of a couple of questions aiming at identifying prior knowledge, e.g. »How do you rate your knowledge about service-oriented computing?«. Answers were measured on a 6-point scale (1 low - 6 high). The *requirements specification* was presented as free-text and had a similar content as the one depicted in Figure 1. However, it contained additional details to better clarify the business semantics that a service should provide. In total, it had a length of 322 words. The first task involved evaluating the descriptions of six services. The *service descriptions* each had a comparable complexity (the WSDL files, e.g., comprised 1342 to 1501 lines of code), but contained varying warehouse management semantics and used differing vocabularies (as e.g. storage or depot instead of warehouse). By varying the vocabulary, we wanted to evaluate whether WS-Functionality is able to better support the assessment of services even in the likely scenario that differing terms for business concepts are used by service providers. Furthermore, none of the service vocabularies conformed to the one used in the requirements specification. However, requirements were constructed in a way that their specification could be matched to corresponding elements of the service descriptions. Using the WSDL documentation feature, we augmented all of the WSDL specifications with

[1] All Materials, including sample solutions, can be requested from the authors.

conventional source code documentation that briefly described the meaning of each function and its parameters (see Figure 2, line 38-39 for an example). In so doing, we wanted to evaluate whether the more detailed WS-Functionality is superior to service descriptions which are enriched with short statements or keywords. Of the services provided, only one completely fulfilled the requirements. Two services did not provide all required operations, since they did not cover a creation of storage/commissioning plans (requirement ❾) or the accounting of stock levels (requirement ❿). The other services varied in their support of requirements ⓿ to ❼. The second task involved comparing the descriptions of two services, whose business semantics varied in four aspects: the maximum of manageable warehouses, a common/customizable storage strategy for warehouse areas, the limit on article counts of storage cells, and their support of a static storage strategy. Again, the service descriptions each had a comparable complexity (WSDL files comprised 1369 to 1422 lines of code). The *post-test* comprised four questions which are depicted in Figure 12. It was conducted to gather information about the satisfaction with the specification language.

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 easy to use?	(not at all)	1 … 2 … 3 … 4 … 5 … 6	(completely)
Q4 – Would you use the service description language again?	(no)	0 ……………………………1	(yes)

Figure 12. Post-test questions on user satisfaction

As mentioned in section 3, we defined usability using the terms effectiveness, efficiency, and satisfaction. Effectiveness can generally be determined by the accuracy with which users achieve their goals (Frøkjær et al. 2000; ISO/IEC 1998). We measured accuracy by giving a score of 1.0 for each correct result and discounting a score of 0.5 for each wrong result. In so doing, we defined a ratio of 1:1 between correct and wrong results as threshold before setting accuracy to cero. This was deemed appropriate since we did not want to count service assessment results which contain more unsuitable than appropriate results as accurate anymore. Efficiency is determined as relation between accuracy and the invested effort (Frøkjær et al. 2000; ISO/IEC 1998). Accordingly we divided the accuracy score of each participant by the time it took him/her to execute the task. The resulting value can be interpreted as a "points per minute" score. Satisfaction was measured by the answers to the post-test depicted in Figure 12.

Based on the collected data, we tested the following propositions which were derived from the more abstract ones stated in section 3: (P1') we expect the treatment group to outperform the control group in matters of effectiveness; (P2') we expect the treatment group to outperform the control group in matters of efficiency; (P3') we expect the treatment group to be more satisfied with the service description than the control group. To identify relevant effects of the treatment, collected data was statistically analyzed and propositions were evaluated on the basis of statistical tests. For the statistical analysis we mostly utilized SPSS (Norusis 2008), where necessary we also applied the programming language and software environment R (Crawley 2007). Due to the fact that accuracy could only take discrete values, we used ordinal logistic regression – the standard method of analysis for discrete response variables (Kutner et al. 2005) – to test our hypothesis concerning effectiveness. We also applied ordinal logistic regression to analyze questions Q1 to Q4 of the post-test, since the respective score has an ordinal scale. As the efficiency score is a continuous value, we performed Kolmogorow-Smirnow and Levene tests to check for normality and variance conditions. Where possible, we then executed a Student's t test. Otherwise the Mann-Whitney-U-Test was used. Although our propositions suggest a one-tailed testing, we nevertheless decided on a two-tailed testing since this provides stricter results. Table 1 depicts the test and summary statistics.

Results show that the treatment group, which used both WSDL and WS-Functionality, achieved a significantly higher score than the control group in both the first and second task. This support for proposition P1' indicates that WS-Functionality allows assessors (i) to better analyze the business semantics with the goal to identify appropriate services that fulfill existing functional requirements and (ii) to better identify differences in the business semantics of alternative service candidates. This impression is also supported by observation. As regards task 1, 10 participants of the treatment group were able to identify the appropriate service. Four additional participants identified the appropriate as well as a similar (but inappropriate) service, while only two participants were unable to identify any service or selected more than two inappropriate candidates. In the control group, only five participants selected the appropriate service. Three of them, however, also selected a second (inappropriate) service. The rest of the group either selected two or more inappropriate services or was unable to identify any service at all. As for task 2, seven participants of the treatment group were able to get a score of 4.0. Seven additional participants had a score of 3.0, while the remaining two had a score of 2.5 and 1.0. In the control group, the highest score – obtained by only one participant – was 3.0. No one was hence able to solve the task completely. Instead, most of the group had a score of

2.0, while three participants were unable to get a score at all. Remarkably, the participants of the treatment group where able to perform well despite the fact that the business vocabularies differed between the services. According to the participants, the explicit definition of business terms (which is part of specifications denoted in WS-Functionality) effectively helped them to match differing vocabularies. Interesting is furthermore, that many participants of the treatment group only used WS-Functionality. As checking for technical compatibility was not demanded and the specification in WS-Functionality was deemed to be sufficient, they did not analyze the WSDL files at all.

Table 1. Test and summary statistics for the conducted laboratory experiment

		test (two-tailed)		Treatment > Control?		Summary statistics				
		Test-value	p-value			Min	Max	Mean	Median	Std.dev.
Effect.	T1: Accuracy	Wald statistics: 10,670	0,001**	✓	Treatm.	0	1	0,75	1	0,37
					Control	0	1	0,22	0	0,36
	T2: Accuracy	Wald statistics: 16,648	0,000***	✓	Treatm.	1	4	3,28	3	0,82
					Control	0	3	1,28	1	0,86
Effic.	T1: Accuracy /Time	Mann-Whitney-U statistics: 34,0	0,000***	✓	Treatm.	0	0,053	0,024	0,024	0,015
					Control	0	0,033	0,005	0	0,009
	T2: Accuracy /Time	T statistics: 4,785	0,000***	✓	Treatm.	0,07	0,6	0,26	0,26	0,13
					Control	0	0,2	0,09	0,09	0,07
Satisfaction	Q1: Information	Wald statistics: 0,531	0,466	✗	Treatm.	0	5	3,81	4,5	1,56
					Control	0	6	4,25	4,5	1,48
	Q2: Conciseness	Wald statistics: 1,877	0,171	✗	Treatm.	0	6	4	4	1,59
					Control	0	6	3,13	3	1,93
	Q3: Ease-of-Use	Wald statistics: 6,81	0,009**	✓	Treatm.	0	6	3,88	4	1,54
					Control	1	6	2,44	2	1,55
	Q4: Adoption	Wald statistics: 0,125	0,723	✗	Treatm.	0	1	0,5	0,5	0,52
					Control	0	1	0,44	0	0,51

Legend: T: Task, Q: Question, Effect.: Effectiveness, Effic.: Efficiency, ***: 0,1% (2-tailed) significance, **: 1% (2-tailed) significance, *: 5% (2-tailed) significance, ^{a}: 10% (2-tailed) significance

The test results also show support for P2'. In fact, the results are even more significant which is due to the fact that the task completion time of the treatment group was less than that of the control group. While for task 1 the average task completion time of the control group was about 48 minutes, the treatment group only needed 35 minutes on average. Considering task 2, the task completion time varied less remarkable (15 versus 14 minutes on average), yet the treatment group still needed less time. The reason for this "discrepancy" seems to be apparent: identifying differences in the business semantics of two services in most cases was done by comparing either WS-Functionality or WSDL specifications line to line. For the control group, this task obviously was less complex than comparing the business semantics of six service candidates with existing functional requirements as required in task 1.

P3', however, seems to be only weakly supported. While WS-Functionality was judged to be better than WSDL in three of the four questions, only the third question (ease of use) showed a significant difference. Reasons for this result seem to be twofold: on the one hand, participants generally found WSDL to be more understandable than traditional programming languages. Since the control group did not know WS-Functionality, it intuitively ranked WSDL against such programming languages and gave it a comparatively high score. Because we decided to create only requirements that could be matched to WSDL constructs, the control group moreover viewed the provided information as to be generally complete. This impression might have been further enforced by the sheer size of the WSDL files. On the other hand, some participants of the treatment group degraded Q2 (conciseness) and Q4 (adoption) because WS-Functionality did not provide a graphical presentation. Especially computer science students would have preferred a diagram over a text-based representation. This finding somehow contradicts our expectations (cf. section 5) and will therefore have to be examined in more detail in future iterations of the design science cycle.

Conclusions and Future Directions

In this paper, we argued to explicitly describe the business semantics that is encapsulated in the implementation of software services. We furthermore demonstrated that such an explicit description enhances the ability of consumers to analyze provided services and helps them to better select appropriate services which fulfill existing functional requirements. To that end, we designed the WS-Functionality specification language and tested it against a state-of-the-art approach to service description in a controlled experiment. With WS-Functionality, we introduced a comparatively lightweight approach to specify the business semantics of software services that integrates into the SOC technology stack. Compared to the language elements of traditional conceptual modeling approaches, WS-Functionality uses a reduced set of concept and relationship types for the description. In contrast to approaches like ARIS or IDEF, which usually make use of a mix of several specification languages, WS-Functionality furthermore introduces a single notation that is free of redundancies and able to cover the entire specification content.

The results of our research have implications for both practice and academia. For practice, they signal the need to not solely rely on technically-oriented descriptions during the assessment and evaluation of software services. Instead, services should also come with a detailed description of their underlying business semantics so that they can be efficiently assessed by service consumers. Because of the complex business logic that is provided by software services, the assessment is often done together by designers and business users in a joint development setting (Krafzig et al. 2005). Even in such interdisciplinary scenarios, however, deriving the implemented business semantics from descriptions of the programming interfaces remains a cumbersome task that likely does not lead to acceptable results. As we have shown that the effectiveness of service assessment can be enhanced by providing explicit descriptions of the implemented business semantics, service consumers should actively demand such information from service providers. For service providers, we have shown how the business semantics of software services can be documented and provided to consumers, e.g. on service exchange platforms. Providing such information on service exchange platforms is likely to create a competitive advantage since consumers generally can better find and analyze services with documented business semantics than others (Overhage and Schlauderer 2010). As the WS-Functionality language builds upon conceptual models to describe the business semantics of services, providers will furthermore be able to reuse results from the conceptual design phase of the development process to document the business semantics of services. Such results could successfully be reused in all case studies that we already conducted and we therefore expect the effort to create specifications using WS-Functionality to be kept within reasonable limits. Nevertheless, we will have to more concretely evaluate the effort that is necessary to create specifications using the WS-Functionality language in future research projects.

For academia, we contributed to the building of a theory that covers the description of the business semantics of software services. With the advent of modern component-based and service-oriented computing paradigms, the software artifacts to be reused become coarser-grained and the implemented business semantics becomes more difficult to assess. Yet, the software engineering community solely focuses on developing theories about how to make explicit and specify additional technical service properties as research agendas and the development of the so-called WS-* specifications impressively demonstrate (Papazoglou et al. 2007; Weerawarana et al. 2005). With the development of our solution concept, we introduced an ontology of services to capture their underlying business semantics with a specific system of concepts. This system of concepts is built from three concept and five basic relationship types and turned out to be effective to document the business semantics of services as shown in the example case. However, the proposed service ontology is just a first step which will have to be actively refined and shaped in future research projects. Due to its set of generic concept and relationship types, both the introduced ontology and the resulting specification language are very compact and per se not limited to a specific application domain. To simplify the usage of the specification language, it is conceivable to allow refining its generic constructs and introducing domain-specific concept and relationship types as constructs of domain-specific language versions. In doing so, we could e.g. define a »produces« function and use it to specify »a company **produces** a car« instead of »a company **does** produce a car«. Besides focusing on a mechanism to create such domain-specific language versions, future research will also need to further determine the level of detail that descriptions of the business semantics should actually have to be effective for the assessment of software services.

Regarding the presented version of WS-Functionality, we will work on further ensuring the external validity of our results. On the one hand, we will therefore conduct additional empirical studies in which we vary the application domain and the type of participants. On the other hand, we plan to perform supplementary analytical evaluations of our concept together with a partner from industry. In parallel, we will further refine and improve the presentation format of our language. Above all, we will have to investigate how to raise the level of user satisfaction. To that end, we will conduct empirical comparisons of our presentation format against descriptions which are not based on conceptual models and denoted in ordinary natural language. While natural language texts will likely not provide the necessary level of non-ambiguity, such comparisons will help us finding out if the usability of our presentation format might have been reduced due to the language restrictions introduced to ensure preciseness. Based on the feedback we received during the evaluation, we furthermore plan to develop a second, graphical presentation format to depict specifications. We will then compare the usability of such a graphical presentation with the existing text-based format. Finally, we will work on creating improved tools to systematically assist during the specification and evaluation of business semantics. Process-models for the specification and evaluation will thereby help providers to specify the business semantics of services and consumers to match it with existing functional requirements. With these activities, we aim at further underpinning the claim that we discussed here: for a successful service assessment and evaluation, we need to know what's in a service.

References

Akerlof, G.A. 1970. "The Market for "Lemons": Quality Uncertainty and the Market Mechanism", *The Quarterly Journal of Economics* (84:3), pp. 488-500.

Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A., and Verma, K. 2005. "Web Service Semantics - WSDL-S, Version 1.0", World Wide Web Consortium.

Anderson, J.R. 1982. "Acquisition of Cognitive Skill", *Psychological Review* (89:4), pp. 369-406.

Batra, D., Hoffer, J.A., and Bostrom, R.P. 1990. "Comparing Representations with Relational and EER Models", *Communications of the ACM* (33:2), pp. 126-139.

Bertalanffy, L.v. 1976. *General System Theory*. New York, NY: George Braziller.

Beugnard, A., Jézéquel, J.-M., Plouzeau, N., and Watkins, D. 1999. "Making Components Contract Aware", *IEEE Computer* (32:7), pp. 38-45.

Bodart, F., Patel, A., Sim, M., and Weber, R. 2001. "Should Optional Properties Be Used in Conceptual Modelling: A Theory and Three Empirical Tests", *Information Systems Research* (12:4), pp. 384-405.

Bunge, M. 1977. *Treatise on Basic Philosophy: Volume 3: Ontology I: The Furniture of the World*. Boston, MA: Reidel.

Chinnici, R., Gudgin, M., Moreau, J.J., and Weerawarana, S. 2004. "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", World Wide Web Consortium.

Crawley, M.J. 2007. *The R Book*. Hoboken, NJ: Wiley.

D'Souza, D.F., and Wills, A.C. 1999. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Upper Saddle River, NJ: Addison-Wesley.

Erl, T. 2005. *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ: Prentice Hall.

Fensel, D. 1998. *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Berlin, Heidelberg: Springer.

Frøkjær, E., Hertzum, M., and Hornbæk, K. 2000. "Measuring Usability: Are Effectiveness, Efficiency, and Satisfaction Really Correlated?", *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, The Hague, The Netherlands: ACM, New York, NY, USA, pp. 345-352.

Gehani, N., and McGettrick, A.T. (eds.) 1986. *Software Specification Techniques*. Wokingham: Addison-Wesley.

Gemino, A., and Wand, Y. 2004. "A Framework for Empirical Evaluation of Conceptual Modeling Techniques", *Requirements Engineering* (9:3), pp. 153-168.

Gómez-Pérez, A., Fernández-López, M., and Corcho, O. 2004. *Ontological Engineering*. Berlin, Heidelberg: Springer.

Hadar, I., and Soffer, P. 2006. "Variations in Conceptual Modeling: Classification and Ontological Analysis", *Journal of the Association for Information Systems* (7:8), pp. 569-593.

Han, J. 1998. "A Comprehensive Interface Definition Framework for Software Components", *Proceedings of the Asia-Pacific Software Engineering Conference*, Taipei, Taiwan: IEEE Computer Society Press, pp. 110-117.

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

IEEE 1998. "IEEE Recommended Practice for Software Requirements Specifications", 830-1998, Institute of Electrical and Electronics Engineers, New York, NY.

ISO/IEC 1998. "Ergonomic Requirements for Office Work With Visual Display Terminals (VDTs) - Part 11: Guidance on Usability ", ISO 9241-11:1998(E), International Organization for Standardization.

ISO/IEC 2001. "Software Engineering - Product Quality - Part 1: Quality Model", ISO/IEC Standard 9126-1, International Organization for Standardization.

Kanigel, R. 1997. *The One Best Way: Frederick Taylor and the Enigma of Efficiency*. New York, NY: Viking Penguin.

Krafzig, D., Banke, K., and Slama, D. 2005. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Upper Saddle River, NJ: Prentice Hall.

Kutner, M.H., Nachtsheim, C.J., Neter, J., and Li, W. 2005. *Applied Linear Statistical Models* (5. ed.). Boston, MA: McGraw-Hill/Irwin.

Linehan, M.H. 2008. "SBVR Use Cases", *Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web,* N. Bassiliades, G. Governatori and A. Paschke (eds.), Orlando, FL: Springer, Berlin, Heidelberg, pp. 182-196.

Lyons, J. 1972. *Structural Semantics: Analysis of Vocabulary of Plato*. Oxford: Blackwell.

Lyons, J. 1977. *Semantics: Volume I*. Cambridge: Cambridge University Press.

Marca, D.A., and McGowan, C.L. 2005. *IDEF0 and SADT: A Modeler's Guide*. Auburndale, MA: OpenPress.

Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., and Sycara, K. 2005. "Bringing Semantics to Web Services: The OWL-S Approach", *Semantic Web Services and Web Process Composition, First International Workshop, SWSWPC 2004, Revised Selected Papers,* J. Cardoso and A.P. Sheth (eds.), San Diego, CA, USA: Springer, Berlin, Heidelberg, pp. 26-42.

Mayer, R.E., and Gallini, J.K. 1990. "When Is an Illustration Worth a Thousand Words", *Journal of Educational Psychology* (82:4), pp. 715-726.

McGovern, J., Sims, O., Jain, A., and Little, M. 2006. *Enterprise Service Oriented Architectures - Concepts, Challenges, Recommendations*. Dordrecht: Springer.

Mili, H., Mili, F., and Mili, A. 1995. "Reusing Software: Issues and Research Directions", *IEEE Transactions on Software Engineering* (21:6), pp. 528-561.

Mylopoulos, J. 1992. "Conceptual Modeling and Telos", in: *Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development,* P. Loucopulos and R. Zicari (eds.). Cambridge: John Wiley & Sons, pp. 49-68.

Natis, Y.V. 2003. "Service-Oriented Architecture Scenario", AV-19-6751, Gartner Inc., Stamford, CT.

Nelson, P. 1970. "Information and Consumer Behavior", *The Journal of Political Economy* (78:2), pp. 311-329.

Norusis, M.J. 2008. *SPSS 16.0 Guide to Data Analysis* (2. ed.). Upper Saddle River, NJ: Prentice Hall.

Olle, T.W., Hagelstein, J., MacDonald, I.G., and Rolland, C. 1991. *Information Systems Methodologies: A Framework for Understanding*. Wokingham: Addison-Wesley.

OMG 2008a. "Semantics of Business Vocabulary and Business Rules (SBVR), v1.0 ", formal/2008-01-02, Object Management Group, Needham, MA.

OMG 2008b. "Service oriented architecture Modeling Language (SoaML)", ad/2008-11-01, Object Management Group, Needham, MA.

OMG 2010. "OMG Systems Modeling Language (SysML), Version 1.2", formal/2010-06-02, Object Management Group, Needham, MA.

Ortner, E., and Schienmann, B. 1996. "Normative Language Approach - A Framework for Understanding", *Proceedings of the 15th International Conference on Conceptual Modeling (ER 1996),* B. Thalheim (ed.), Cottbus, Germany: Springer, Berlin, Heidelberg, pp. 261-276.

Overhage, S., and Schlauderer, S. 2010. "The Market for Services: Economic Criteria, Immaturities, and Critical Success Factors", *Proceedings of the 43rd Hawaii International Conference on System Sciences,* IEEE (ed.), Koloa, Kauai, HI: IEEE Computer Society Press, Los Alamitos, CA, pp. 1-10.

Papazoglou, M.P. 2007. *Web Services: Principles and Technology*. Upper Saddle River, NJ: Prentice Hall.

Papazoglou, M.P., Traverso, P., Dustdar, S., and Leymann, F. 2007. "Service-Oriented Computing: State of the Art and Research Challenges", *IEEE Computer* (40:11), pp. 38-45.

Prieto-Díaz, R. 2003. "A Faceted Approach to Building Ontologies", *Proceedings of the IEEE International Conference on Information Reuse and Integration IRI 2003*, Las Vegas, NV, USA: IEEE Systems, Man, and Cybernetics Society, pp. 458-465.

Scheer, A.W. 2000. *ARIS - Business Process Frameworks* (3. ed.). Berlin, Heidelberg: Springer.

Speed, J., Councill, W.T., and Heineman, G.T. 2001. "Component-Based Software Engineering as a Unique Engineering Discipline", in: *Component-Based Software Engineering: Putting the Pieces Together,* W.T. Councill and G.T. Heineman (eds.). Upper Saddle River, NJ: Addison-Wesley, pp. 673-691.

Takeda, H., Veerkamp, P., Tomiyama, T., and Yoshikawa, H. 1990. "Modeling Design Processes", *AI Magazine* (11:4), pp. 37-48.

Topi, H., and Ramesh, V. 2002. "Human Factors Research on Data Modeling: A Review of Prior Research, an Extended Framework and Future Research Directions", *Journal of Database Management* (13:2), pp. 3-15.

van der Aalst, W.M.P., and Kumar, A. 2003. "XML-Based Schema Definition for Support of Interorganizational Workflows", *Information Systems Research* (14:1), pp. 23-46.

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

Vessey, I., and Conger, S.A. 1994. "Requirements Specification: Learning Object, Process, and Data Methodologies", *Communications of the ACM* (37:5), pp. 102-113.

Vitharana, P., Zahedi, F., and Jain, H. 2003. "Knowledge-Based Repository Scheme for Storing and Retrieving Business Components: A Theoretical Design and an Empirical Analysis", *IEEE Transactions on Software Engineering* (29:7), pp. 649-664.

Wand, Y., and Weber, R. 1993. "On the Ontological Expressiveness of Information Systems Analysis and Design Grammars", *Journal of Information Systems* (3:4), pp. 217-237.

Weerawarana, S., Curbera, F., Leymann, F., Ferguson, D.F., and Storey, T. 2005. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Upper Saddle River, NJ: Prentice Hall.

Weyuker, E.J. 1998. "Testing Component-Based Software: A Cautionary Tale", *IEEE Software* (15:5), pp. 54-59.

Wirth, N. 1971. "Program Development by Stepwise Refinement", *Communications of the ACM* (14:4), pp. 221-227.